Classical Logic and the Curry–Howard Correspondence

Finn Lawler
Foundations and Methods Group
flawler@cs.tcd.ie

October 23, 2008

Abstract

We present the Curry–Howard correspondence for constructive logic via natural deduction, typed λ-calculus and cartesian closed categories. We then examine how the correspondence may be extended to classical logic and non-constructive proofs, and discuss some of the problems and questions that then arise.

Contents

1 Constructive Logic

1.1 Introduction

The question of the constructiveness of proofs lies at the intersection of logic, computer science and the philosophy of mathematics. In philosophy, we ask for the meaning of an existential formula, that is, one of the form $\exists x.Fx$. In particular, we ask what a proof of such a formula consists of. Constructivists argue that such a proof must construct an object a such that Fa, while classically it is enough to prove that it is impossible for no such a to exist. Constructivists must for this reason reject certain principles of classical logic, such as the excluded middle $P \vee \neg P$ and the double-negation rule $\neg\neg P \rightarrow P$. Constructive logic, therefore, is a proper subsystem of classical logic, and an object worthy of study in its own right.

In computer science, the Curry–Howard correspondence, and generalizations[1] of it known as 'realizability interpretations', give a remarkably elegant way to extract from a constructive formal proof an algorithm that justifies that proof. In particular, a proof of an existential theorem $\exists x.Fx$ yields an algorithm that computes a witness for that theorem — namely an object a such that Fa. This means, for example, that a formula like $\forall \vec{x}.(F(\vec{x}) \rightarrow \exists y.G(\vec{x}, y))$ can be interpreted as a specification, with the predicate F as precondition and G as postcondition. A constructive formal proof of this formula will yield a program that, given (representations of) objects $\vec{a}$ satisfying $F(\vec{a})$ (together with a proof of this fact) will return an object b such that (and a proof that) $G(\vec{a}, b)$.

For a long time, this correspondence between constructive proofs and algorithms was thought to bolster the position of the constructivists. While constructive proofs had a concrete meaning expressed in terms of computations, it seemed that classical proofs did not, and thus that their meaning was unclear. This picture has changed in the last fifteen years or so with the discovery of a Curry–Howard correspondence for classical logic, which is the subject of this report.

To begin, we will present the Curry–Howard correspondence for constructive (minimal and intuitionistic) logic, expressed in terms of natural deduction. There follows a discussion of intuitionistic sequent calculus LJ and two term calculi for it, which leads, via a natural generalization, to a consideration of classical sequent calculus LK. We then present various equivalent term calculi for LK, and conclude with an examination of the outstanding problems and issues with this interpretation of classical logic. In particular, we will see that while there is a very neat three-legged correspondence linking minimal logic, typed λ-calculi and cartesian closed categories, and while there are λ-calculi that correspond closely to classical logic, the problem of giving sufficiently general categorical models of classical logic is very much open. This is closely related to the familiar problem of giving an appropriate notion of equality for non-deterministic rewrite systems.

Sections 1.5 and 2.5 require some knowledge of basic category theory. See e.g. [Mac98] (the standard reference) or [BW95].

We conclude this section with some notation and basic definitions.

1.1.1 Definition. The language of propositional logic is generated by the following grammar, where P ranges over a countably infinite set of proposition letters $\mathcal{P} = \{p_1, p_2, \ldots\}$.

$$F ::= P \mid \bot \mid F \rightarrow F \mid F \wedge F \mid F \vee F \mid \neg F$$

Strings of the language are called *propositional formulas*, or just *formulas*. The four connectives bind increasingly tightly from left to right, so that e.g. $A \rightarrow \neg B \wedge C$ is parsed as $A \rightarrow ((\neg B) \wedge C)$. The connectives $\wedge, \vee$ are associative, and $\rightarrow$ associates to the right.

[1] Actually, realizability was formulated nearly twenty-five years before the Curry–Howard correspondence became widely known — 1945 [Kle45] versus 1969 [How80] — and over a decade before the latter's first appearance in print in 1958 [CF68].

The language of implicative propositional logic is that generated by the first three clauses. The last clause may be eliminated by defining $\neg F$ as $F \to \bot$. The metavariables $A, B, C, \ldots$ will be used to range over formulas.

1.1.2 Definition. A morphism $f : A \to B$ between objects A and B is an *isomorphism* if there exists a morphism $\tilde{f} : B \to A$ such that $\tilde{f}$ is a left and right inverse to f, i.e. $\tilde{f} \circ f = 1_A$ and $f \circ \tilde{f} = 1_B$, where 1 is the identity morphism. We write $f : A \cong B$ when this holds. In general, the inverse to an isomorphism f will be denoted $\tilde{f}$ (rather than f^{-1}).

1.1.3 Definition. A *partial order* is a relation $\leq$ on a set S that satisfies the following properties for all $x, y, z \in S$.

- Reflexivity:
$$x \leq x$$
- Transitivity
$$x \leq y \text{ and } y \leq z \text{ implies } x \leq z$$
- Antisymmetry
$$x \leq y \text{ and } y \leq x \text{ implies } x = y$$

A structure $\langle S, \leq_S \rangle$ where S is a set and $\leq_S$ a partial order is called a *poset*, and is usually referred to simply as S. A homomorphism between posets S and T, i.e. a function $f : S \to T$ that respects the orders, is called a *monotone function.*

A *preorder* is a relation $\sqsubseteq$ that is reflexive and transitive. A preordered set can be viewed as a category where each hom set has at most one element, with $\hom(a, b) = \{*\}$ iff $a \sqsubseteq b$. Monotone functions are then simply functors.

1.1.4 Definition. A *Galois connection* is given by two posets S and T together with two monotone functions $f^* : S \to T$ (*lower adjoint*) and $f_* : T \to S$ (*upper adjoint*) such that for all $s \in S, t \in T$

$$f^*(s) \leq_T t \text{ iff } s \leq_S f_*(t)$$

or equivalently

$$s \leq_S (f_* \circ f^*)(s) \text{ and } (f^* \circ f_*)(t) \leq_T t$$

When S and T are viewed as categories as above, this states exactly that $f^* \dashv f_*$.

1.1.5 Definition. Given sets S, T of trees, a *strong isomorphism* $\phi : S \equiv T$ is a bijection $\phi : S \cong T$ such that for all $s \in S$, $s \cong \phi(s)$ (and thus $\tilde{\phi}(t) \cong t$ for $t \in T$). If such an isomorphism exists, we say that S and T are *strongly isomorphic.*

This terminology is not standard.

1.2 The BHK Interpretation

Although several varieties of constructivism were promulgated during the twentieth century, we will consider only the most influential — the school of intuitionism, founded by Brouwer. Intuitionism had its roots (see [AR01]) in the late-nineteenth-century reaction, exemplified by Kronecker, against the increasing use of transfinite, non-constructive methods of proof. Just as Kronecker, according to folklore, had declared that

God made the integers — all the rest is the work of man,

Brouwer developed his philosophy as an extension of Kant's characterization of the natural numbers as given by the 'forms of intuition', and thus primordial. The rest of mathematics was to be reconstructed on this basis using purely constructive techniques (see e.g. [Dum00]).

Brouwer's doctrines, and the work by Heyting and Kolmogorov [Kol67] on intuitionistic logic, gave rise to a constructive interpretation of the logical connectives that has become standard. It is known as the Brouwer–Heyting–Kolmogorov (BHK) interpretation, and spells out inductively the conditions intuitionistically necessary for an object to be considered a proof of or to 'realize' a given formula (see e.g. [Hey66]):

- The form of a proof of an atomic formula is taken as known. For example, in Heyting arithmetic, a direct computation will serve as a proof of $t = u$.
- A proof of $A \wedge B$ is a pair $\langle a, b \rangle$ consisting of a proof a of A and a proof b of B.
- A proof of $A \vee B$ is a either a proof of A or a proof of B, together with an indication of which it is (i.e. disjoint union).
- A proof of $A \rightarrow B$ is a 'construction' c that turns any proof a of A into a proof $c(a)$ of B.
- There is no proof of $\perp$.

These notions of ordered pair, union etc. should be familiar to any computer scientist. Notice that the clause for implication means that a system of realizers for intuitionistic logic must include higher-order functions.

In predicate logic, the quantifiers are interpreted as follows, where S is one of a set of given 'sorts' (and usually omitted in the one-sort case).

- A proof of $\forall x : S.A[x]$ is a construction c that turns any object $s \in S$ into a proof $c(s)$ of $A[s]$.
- A proof of $\exists x : S.A[x]$ is a pair $\langle s, p(s) \rangle$ consisting of an object $s \in S$ and a proof $p(s)$ of $A[s]$.

From this one can see that, as mentioned above, a constructive proof of a Π_2 formula $\forall x.(F(x) \rightarrow \exists y.G(x, y))$ must take an object a and a proof of $F(a)$ and return a pair containing an object b and a proof of $G(a, b)$.

This interpretation of the connectives and quantifiers is obviously useful for deriving provably correct programs. A notable example is the proof assistant Coq[2], which, as well as helping the user to derive constructive formal proofs, can extract the algorithmic content of those proofs, yielding programs in e.g. ML or Haskell.[3]

[2] http://coq.inria.fr

[3] One particularly impressive case is the CompCert compiler — a C compiler for MacOS X entirely verified in Coq. See http://compcert.inria.fr.

1.3 Natural Deduction

In the past, formal proofs were usually given in the traditional linear style due (or attributed) to Euclid, where a derivation would begin with axioms and proceed line by line, with each line derived from those preceding it by means of some inference rule. Nowadays such logics are known as 'Hilbert systems'. This format can be somewhat cumbersome and inelegant, both because it does not follow the reasoning-patterns of ordinary mathematics and because it is itself not especially suited to mathematization.

In the 1930s, the logician Gerhard Gentzen introduced[4] a formalism called 'natural deduction' [Gen69], which instead of axioms uses a tree-like layout that proceeds from hypotheses to conclusion via inference rules, one or more for each connective. These systems are more elegant and better suited to the needs of proof theory than Hilbert systems.

1.3.1 Definition (Adapted from [TS00])**.** A *natural deduction derivation* is a finite tree, where leaves are labelled with a formula, a name and a *discharged* flag, and nodes are labelled with the name of an inference rule and a formula (the *conclusion* of the rule). The conclusion of a leaf with formula A is A. Derivations are thus generated by the following grammar, where x ranges over a countable set of names, ρ over a set of inference rules and A over formulas.

$$D ::= A^{x} \;\mid\; \overline{A}^{x} \;\mid\; \dfrac{D \quad \cdots \quad D}{A}\,\rho$$

The overlining of the second form indicates that the discharged flag is set. The leaves of derivations are intended to represent hypotheses, each having a name x, written as A^x or $x : A$. The sets of leaves with the same names form *packets* of hypotheses, and distinct formulas must belong to distinct packets. Packets may be empty. Inference rules may discharge packets of hypotheses, meaning that the proof no longer depends on the formula involved. Each leaf belonging to the packet then has its discharged flag set. A *closed* derivation is one where every packet has been discharged. An undischarged packet is *open* or *active.*

In everyday mathematical practice, one reasons by beginning with a hypothesis A, deducing some consequence B and concluding $A \to B$, discharging a hypothesis in the process. Symmetrically, if one has proved $A \to B$ and A, one can conclude B, as with *modus ponens.*

1.3.2 Definition. The system $NM^{\to}$ of implicative minimal propositional logic is a system of natural deduction given by the following rules.

$$\begin{array}{c} \overline{A}^{x} \\ \vdots \\ \dfrac{B}{A \to B}\,{\to}^{+}_{x} \end{array} \qquad\qquad \dfrac{A \to B \quad A}{B}\,{\to}^{-}$$

In the first rule, the packet A^x is discharged.

[4] Gentzen's system was anticipated by Jáskowski — see [Pra06, Appendix C] for a brief overview.

Observe that the rule $\to^+$ introduces the connective $\to$ while the rule $\to^-$ eliminates it. This pattern is followed by the natural deduction rules for the other connectives, and is closely analogous to the constructor/destructor distinction familiar to computer scientists.

1.3.3 Remark. As an example, one of the standard axioms for Hilbert systems can be proved in $NM^{\to}$ with the following closed derivation.

$$\cfrac{\cfrac{\cfrac{}{A}\,x}{B \to A}\,\to^+_y}{A \to B \to A}\,\to^+_x$$

This derivation makes use of an empty packet B^y. A proof of another of the standard axioms uses $\to^-$:

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{}{A \to B \to C}\,x \quad \cfrac{}{A}\,z}{B \to C}\,\to^- \qquad \cfrac{\cfrac{}{A \to B}\,y \quad \cfrac{}{A}\,z}{B}\,\to^-}{C}\,\to^-}{A \to C}\,\to^+_z}{(A \to B) \to (A \to C)}\,\to^+_y}{(A \to B \to C) \to (A \to B) \to (A \to C)}\,\to^+_x$$

This derivation is also closed. If the last rule was erased, the packet x would remain open.

1.3.4 Definition. The rules for conjunction $\wedge$ are these, where $i = 1, 2$:

$$\frac{A_1 \qquad A_2}{A_1 \wedge A_2}\,\wedge^+ \qquad\qquad \frac{A_1 \wedge A_2}{A_i}\,\wedge^-_i$$

The system $NM^{\to}$ extended with these rules is called $NM^{\to\wedge}$.

1.3.5 Definition. The rules for disjunction are as follows, where $i = 1, 2$:

$$\frac{A_i}{A_1 \vee A_2}\,\vee^+_i \qquad\qquad \frac{A_1 \vee A_2 \qquad \begin{array}{c}\overline{A_1}^{\,x} \\ \vdots \\ B\end{array} \qquad \begin{array}{c}\overline{A_2}^{\,y} \\ \vdots \\ B\end{array}}{B}\,\vee^-_{x,y}$$

In the second rule, the hypotheses x and y are discharged. The system $NM^{\to\wedge}$ extended with these rules is the full system NM.

1.3.6 Definition. Intuitionistic natural deduction NJ is obtained from NM by adding the following rule of *ex falso quodlibet*:

$$\frac{\perp}{A}\,\perp^-$$

Classical natural deduction NK adds to NJ the rule of *duplex negatio affirmat* (double negation):

$$\frac{\neg\neg A}{A}\,\neg\neg^-$$

An alternative to NK is the system NK_m of *multiple-conclusion natural deduction* (see e.g. [Dum00]), where nodes are labelled with a sequence of formulas. This system will be treated in section 2.4.

1.3.7 Definition. We write $\vdash_L A$ to mean that the judgement A can be derived in the system L. If L allows hypotheses, then we write $\Gamma \vdash_L A$ when A can be derived under the hypotheses Γ. If D is a derivation of A under Γ then we write $D :: \Gamma \vdash_L A$. The name of a system L may be used to denote the set of derivations of L.

An alternative to the above treatment of hypotheses is to maintain a list of those active at each stage of the proof. This is more cumbersome for humans, but better for theorem provers, and makes induction on proof-trees easier. It also makes empty hypothesis-packets explicit.

1.3.8 Definition. A *formula context* is a finite set of named formulas $\{x_1 : A_1, \ldots, x_n : A_n\}$, where each of the x_i are distinct. The set-braces are usually omitted. A (minimal) *sequent* is a string of the form $\Gamma \triangleright A$, where Γ is a formula context. The union of two formula contexts Γ and Δ is denoted Γ, Δ.

The definition of a derivation is modified accordingly.

1.3.9 Definition. A *natural deduction derivation with explicit hypotheses* is a finite sequent-labelled tree where leaves are of the form

$$\Gamma, x : A \triangleright A$$

and nodes are well-formed with respect to a set of inference rules.

1.3.10 Definition. The system NM may be formulated with explicit hypotheses using the following rules, where the Ax rule gives the form of leaf sequents as above.

$$\frac{}{\Gamma, x : A \triangleright A}\,\mathrm{Ax}$$

$$\frac{\Gamma, x : A \triangleright B}{\Gamma \triangleright A \to B}\,{\to}^+_x \qquad \frac{\Gamma \triangleright A \to B \qquad \Gamma \triangleright A}{\Gamma \triangleright B}\,{\to}^-$$

$$\frac{\Gamma \triangleright A \qquad \Gamma \triangleright B}{\Gamma \triangleright A \wedge B} \qquad \frac{\Gamma \triangleright A_1 \wedge A_2}{\Gamma \triangleright A_i}\,\wedge^-_i$$

$$\frac{\Gamma \triangleright A_i}{\Gamma \triangleright A_1 \vee A_2}\,\vee^+_i \qquad \frac{\Gamma \triangleright A_1 \vee A_2 \qquad \Gamma, x : A_1 \triangleright B \qquad \Gamma, y : A_2 \triangleright B}{\Gamma \triangleright B}\,\vee^-_{x,y}$$

1.3.11 Remark. Let NM' denote the above system: clearly, $\Gamma \vdash_{NM} A$ iff $\vdash_{NM'} \Gamma \triangleright A$. From now on, we use NM' instead of NM, regarding the two as equivalent. We extend the turnstile notation to systems with explicit hypotheses by writing $\Gamma \vdash A$ instead of $\vdash \Gamma \triangleright A$.

1.4 λ-Calculus

The λ-calculus is an abstract theory of functions, on which many modern programming languages are ultimately based. Because it axiomatizes the notion of 'computable higher-order function', it is well suited to interpreting constructive logic.

We begin with some standard definitions (see e.g. [Bar81, HS86]):

1.4.1 Definition. The set Λ of λ-terms M is given by the grammar:

$$M \ ::= \ x \mid \lambda x.M \mid (M\ M)$$

where x ranges over a countably infinite set $\mathcal{V}$ of variables. Application $(M\ N)$ is left associative, so that e.g. the term $((M\ N)\ L)$ may be written MNL. The scope of a λ extends as far to the right as possible, until interrupted by a close-parenthesis or a space.

1.4.2 Definition. The set $\mathrm{FV}(M)$ of free variables of a λ-term M is defined as follows:

$$\begin{array}{rcl} \mathrm{FV}(x) & = & \{x\} \\ \mathrm{FV}(\lambda x.M) & = & \mathrm{FV}(M) \setminus \{x\} \\ \mathrm{FV}((M\ N)) & = & \mathrm{FV}(M) \cup \mathrm{FV}(N) \end{array}$$

The set $\mathrm{BV}(M)$ of bound variables of M is defined as

$$\begin{array}{rcl} \mathrm{BV}(x) & = & \varnothing \\ \mathrm{BV}(\lambda x.M) & = & \mathrm{BV}(M) \cup \{x\} \\ \mathrm{BV}((M\ N)) & = & \mathrm{BV}(M) \cup \mathrm{BV}(N) \end{array}$$

1.4.3 Definition. The substitution $M[x := N]$ of a λ-term N for a variable x in a λ-term M is defined as follows:

$$\begin{array}{rcl} x[x := N] & = & N \\ y[x := N] & = & y \\ (M_1\ M_2)[x := N] & = & (M_1[x := N]\ M_2[x := N]) \\ (\lambda y.M)[x := N] & = & \lambda y'.(M[y := y'][x := N]),\ y' \notin \mathrm{FV}(N) \end{array}$$

1.4.4 Definition. Two λ-terms M, N are said to be *α-equivalent* $M \equiv_\alpha N$ if they differ only in the names of their bound variables. Axiomatically,

$$\lambda x.M \equiv_\alpha \lambda y.(M[x := y])$$

and if $=$ denotes syntactic equality, then α-equivalence is the quotient $=/\equiv_\alpha$. Unless otherwise specified, we consider terms only up to this relation.

1.4.5 Definition. The (one-step) relation of *β-reduction* $\succ_\beta$ is defined on Λ as

$$(\lambda x.M\ N) \ \succ_\beta \ M[x := N]$$

β-reduction $\succeq_\beta$ is defined as the reflexive and transitive closure of $\succ_\beta$, and β-equality $=_\beta$ as its reflexive, transitive, symmetric closure.

The relation of *η-reduction* is defined as

$$\lambda x.(Mx) \ \succ_\eta \ M$$

where $x \notin \mathrm{FV}(M)$. As with β-reduction, there is a reflexive and transitive closure $\succeq_\eta$ and a symmetric closure $=_\eta$ of that.

A term M such that $M \succ M'$ is called a *reducible expression*, or *redex*, and the term M' is its *reduct*. A term that contains no redexes is said to be in *normal form*.

These six relations are extended to the whole of Λ by congruence with respect to the term constructors, so that e.g. if $M \succeq M'$ then $\lambda x.M \succeq \lambda x.M'$ and if also $N \succeq N'$ then $MN \succeq M'N'$. The unions of these relations will be denoted $\succ_{\beta\eta}$, $\succeq_{\beta\eta}$, $=_{\beta\eta}$, etc. The second is a preorder, the third an equivalence relation. The subscript may be dropped if there is no danger of ambiguity.

The operations of λ-abstraction and application provide higher-order functions to interpret implication. We may also add constructs to interpret conjunction and disjunction, namely ordered pairs and unions.

1.4.6 Definition. Pairing and union operators are added to the λ-calculus as follows: if M, N are terms then

$$\pi_1 M \qquad \pi_2 M \qquad \langle M, N\rangle$$

are terms, with reduction rules

$$\begin{array}{rcl} \pi_1\langle M, N\rangle & \succ_\beta & M \\ \pi_2\langle M, N\rangle & \succ_\beta & N \\ \\ \langle \pi_1 M, \pi_2 M\rangle & \succ_\eta & M \end{array}$$

Unions are defined using injection and case operators:

$$\iota_1 M \qquad \iota_2 M \qquad [x.M_1|y.M_2]N$$

with reduction rules

$$\begin{array}{rcl} [x.M_1|y.M_2]\iota_1 N & \succ_\beta & M_1[x := N] \\ [x.M_1|y.M_2]\iota_2 N & \succ_\beta & M_2[y := N] \\ \\ [x.\iota_1 x|y.\iota_2 y]M & \succ_\eta & M \end{array}$$

1.4.7 Remark. The η-reduction rules are sometimes interpreted as *increasing*, e.g. $M \succeq \langle \pi_1 M, \pi_2 M\rangle$. We may then think of the projections as an ordered pair $(\pi_1, \pi_2) : \Lambda \to \Lambda \times \Lambda$, in which case we have, for all M, N:

$$\begin{array}{rcl} ((\pi_1, \pi_2) \circ \langle -, -\rangle)(M, N) & \succeq & (M, N) \\ M & \succeq & (\langle -, -\rangle \circ (\pi_1, \pi_2))(M) \end{array}$$

which expresses a Galois connection (see def. 1.1.4) between the pairing and projection operators. A similar relation holds between the term constructors for the other types, when η is interpreted as an expansion. Developing this point of view leads to the use of bicategories and lax adjunctions to model typed rewrite systems: see e.g. [See87, Gha95, Hil96].

Types

The λ-calculus with pairing and unions is a system of realizers for NM. In order to restrict the set of λ-terms to those that realize theorems of NM, we introduce types. Typing is of course also used in programming languages to provide compile-time consistency-checking.

1.4.8 Definition. The set $\mathcal{T}(S)$ of simple types A is freely generated over a set S of atomic types by way of the following grammar, where $P \in S$:

$$A ::= P \mid \bot \mid A \to A \mid A \times A \mid A + A$$

The set $\mathcal{T}^{\to}(S)$ is that generated by the first three productions, and similarly for other superscripts. The set S will usually be left implicit.

1.4.9 Definition. The set $\Lambda_{\mathcal{T}^{\to}}$ of simply-typed λ-terms is a subset of $\Lambda \times \mathcal{T}^{\to}$ determined by derivability in the system $\lambda^{\to}$, defined below. An analogous notation may be used for extensions of the system to include other types defined above. In symbols:

$$\Lambda_{\mathcal{T}^{\to}} \;=\; \{(M, A) \in \Lambda \times \mathcal{T}^{\to} \mid \exists \Gamma . \Gamma \vdash_{\lambda^{\to}} M : A\}$$

The functions FV and BV are defined just as for Λ, giving the free and bound variables of a term together with their types.

A term M of type A is denoted $M : A$ or M^A, although terms will often have most or all of their types omitted.

1.4.10 Definition. A *type environment* is a set of variable–type pairs, where the variables are distinct. They are written as sequences:

$$\Gamma \;::=\; \cdot \mid \Gamma, x : A$$

Given type environments Γ and Δ, their union is denoted Γ, Δ. The pair $x : A$ may stand for the obvious singleton environment.

Type environments may be regarded as functions $\mathcal{V} \to \mathcal{T}$, so that e.g. if $\Gamma = x_1 : A_1, \ldots x_n : A_n$, then $\mathrm{dom}(\Gamma) = \{x_1, \ldots, x_n\}$, and so on.

1.4.11 Definition. The simply-typed λ-calculus $\lambda^{\to}$ is defined by the following rules, where Γ is a type environment:

$$\frac{}{\Gamma, x : A \triangleright x : A}$$

$$\frac{\Gamma, x : A \triangleright M : B}{\Gamma \triangleright \lambda x . M : A \to B}\,{\to}^{+} \qquad \frac{\Gamma \triangleright M : A \to B \qquad \Gamma \triangleright N : A}{\Gamma \triangleright (M\ N) : B}\,{\to}^{-}$$

Terms of product type are derived as follows, for $i = 1, 2$:

$$\frac{\Gamma \triangleright M_1 : A_1 \qquad \Gamma \triangleright M_2 : A_2}{\Gamma \triangleright \langle M_1, M_2 \rangle : A_1 \times A_2}\,\times^{+} \qquad \frac{\Gamma \triangleright M : A_1 \times A_2}{\Gamma \triangleright \pi_i M : A_i}\,\times_i^{-}$$

Terms of sum type are derived as follows, for $i = 1, 2$:

$$\frac{\Gamma \triangleright M : A_i}{\Gamma \triangleright \iota_i M : A_1 + A_2}\,+_i^{+}$$

$$\frac{\Gamma \triangleright M : A_1 + A_2 \qquad \Gamma, x : A_1 \triangleright N_1 : B \qquad \Gamma, y : A_2 \triangleright N_2 : B}{\Gamma \triangleright [x.N_1 | y.N_2] M : C}\,+^{-}$$

The Curry–Howard Correspondence

1.4.12 Remark. Observe now that there is a bijection between proofs of $NM^{\to}$ and derivations in $\lambda^{\to}$. We map the latter to the former with a function that forgets the λ-term:

$$\Gamma \triangleright M : A \quad \mapsto \quad \Gamma \triangleright A$$

which obviously commutes with the inference rules, while from an NM-proof we can build a typing derivation by induction:

$$\frac{}{\Gamma, x : A \rhd A} \qquad \mapsto \qquad \frac{}{\Gamma, x : A \rhd x : A}$$

$$\frac{\Gamma, x : A \rhd B}{\Gamma \rhd A \to B} \qquad \mapsto \qquad \frac{\Gamma, x : A \rhd M : B}{\Gamma \rhd \lambda x.M : A \to B}$$

$$\frac{\Gamma \rhd A \to B \qquad \Gamma \rhd A}{\Gamma \rhd B} \qquad \mapsto \qquad \frac{\Gamma \rhd M : A \to B \qquad \Gamma \rhd N : A}{\Gamma \rhd MN : B}$$

It is easy to see that this is a bijection — indeed, an $NM^{\to}$ proof and a $\lambda^{\to}$ derivation that map to each other are themselves isomorphic as trees, and hence we get a strong isomorphism (def. 1.1.5) $NM^{\to} \equiv \lambda^{\to}$. This extends in the obvious way to the full system NM, by the mapping on types $\times \leftrightarrow \wedge, + \leftrightarrow \vee$.

In fact, we can go further than this, since a typed λ-term is isomorphic to its own typing derivation. The only snag in constructing a bijection is that a type environment may contain unused variables. However, we can single out a minimal environment, which is the point of the following.

1.4.13 Proposition. *Define a partial order on $\lambda^{\to}$ derivations $D :: \Gamma \rhd M : A$ and $D' :: \Gamma' \rhd M : A$ by $D \leq D'$ iff $\Gamma \subseteq \Gamma'$. Then there are functions*

$$\begin{tikzcd} \lambda^{\to} \arrow[r, shift left, "\phi"] & \Lambda_{\mathcal{T}^{\to}} \arrow[l, shift left, "\psi"] \end{tikzcd}$$

such that $\phi\psi = 1_{\Lambda_{\mathcal{T}^{\to}}}$ and $\psi\phi \leq 1_{\lambda^{\to}}$ in the pointwise order.

Proof. First note that if $\Gamma \vdash M : A$ then $\mathrm{FV}(M) \subseteq \Gamma$. Define $\phi(D :: \Gamma \rhd M : A) = M$, and let ψ_Γ be defined as follows, where Γ is a type environment:

$$\psi_\Gamma(x : A) \quad = \quad \frac{}{\Gamma \rhd x : A}$$

$$\psi_\Gamma(\lambda x^A.M^B) \quad = \quad \frac{\psi_{\Gamma, x:A}(M)}{\Gamma \rhd \lambda x.M : A \to B}$$

$$\psi_\Gamma(M^{A \to B} N^A) \quad = \quad \frac{\psi_\Gamma(M) \qquad \psi_\Gamma(N)}{\Gamma \rhd MN : B}$$

Define $\psi(M) = \psi_{\mathrm{FV}(M)}(M)$, and observe that if $\psi(M) = D :: \Gamma \rhd M : A$ then $\Gamma = \mathrm{FV}(M)$ (thus Γ is the 'minimal environment' we are looking for). Observe also that the value of $\psi_\Gamma(x : A)$ is a well-formed axiom sequent, because if M is the term we started with then either $x : A \in \mathrm{FV}(M)$ or $x : A \in \mathrm{BV}(M)$: in the latter case $x : A \in \Gamma$ by the second clause.

Now for some D and Γ

$$\phi\psi(M) = \phi(D :: \Gamma \rhd M : A) = M$$

and thus $\phi\psi = 1_{\Lambda_{\mathcal{T}^{\to}}}$. Given $D :: \Gamma \rhd M : A$

$$\psi\phi(D) = \psi_{\mathrm{FV}(M)} M = D' :: \Gamma' \rhd M : A$$

where $\Gamma' = \mathrm{FV}(M)$. Since $\mathrm{FV}(M) \subseteq \Gamma$, we have that $\Gamma' \subseteq \Gamma$, implying $D' \leq D$ and thus that $\psi\phi \leq 1_{\lambda^{\to}}$. □

Again, this extends to the systems with the full complement of types or connectives. By considering derivations up to the (practically trivial) equivalence relation $\sim\ = \ker\ \psi\phi$ that disregards unused variables and extending it to NM via the bijection of remark 1.4.12, we get the strong isomorphisms:

$$NM/\sim \quad \equiv \quad \lambda^{\to\times+}/\sim \quad \equiv \Lambda_{\mathcal{T}}$$

So to any NM proof there corresponds an isomorphic typed λ-term — but the correspondence is even closer than that, extending to cover reduction rules too, as the following shows.

1.4.14 Proposition (Subject Reduction)**.** *If* $\Gamma \vdash_{\lambda^\to} M : A$*, and* $M \succeq M'$*, then* $\Gamma \vdash_{\lambda^\to} M' : A$*.*

Proof. We show that the reduction relation on Λ can be extended to $\lambda^\to$, thereby yielding a derivation of $\Gamma \ \rhd\ M' : A$. Indeed, given the equivalences shown above, it should not be surprising that this can be done: we could simply use the functions ϕ and ψ from the previous proposition. However, we will instead give the rewrite rules explicitly.

We define substitution of derivations using the following meta-rule:

$$\dfrac{\Gamma \rhd N : A \qquad \Gamma, x : A \rhd M : B}{\Gamma \rhd M[x := N] : B}\ \mathrm{Cut}_x$$

The definition of substitution (def. 1.4.3) is extended via Cut to derivations in $\lambda^\to$ as shown in figure 1.4.1. Now we can lift $\succeq_{\beta\eta}$ to $\lambda^\to$:

$$\dfrac{\dfrac{\overset{D}{\Gamma, x : A \rhd M : B}}{\Gamma \rhd \lambda x.M : A \to B} \qquad \overset{E}{\Gamma \rhd N : A}}{\Gamma \rhd \lambda x.M\ N : B} \quad \succeq_\beta \quad \dfrac{\overset{E}{\Gamma \rhd N : A} \qquad \overset{D}{\Gamma, x : A \rhd M : B}}{\Gamma \rhd M[x := N] : B}$$

$$\begin{array}{c}\dfrac{\dfrac{\overset{D}{\Gamma \rhd M : A \to B} \qquad \dfrac{}{\Gamma, x : A \rhd x : A}}{\Gamma, x : A \rhd Mx : B}}{\Gamma \rhd \lambda x.Mx : A \to B} \\ (x \notin \mathrm{FV}(M))\end{array} \quad \succeq_\eta \quad \overset{D}{\Gamma \rhd M : A \to B}$$

We extend the reduction relation to $\lambda^\to$ by congruence, as with Λ (def. 1.4.5). Induction on derivations then shows that given $D :: \Gamma \ \rhd\ M : A$, if $M \succeq M'$ then there is a D' such that $D \succeq D'$ and $D' :: \Gamma \rhd M' : A$. □

The above shows that for any NM proof of a formula A there is a typed λ term isomorphic to it, which, in the spirit of the BHK interpretation, may be thought of as a *realizer* of A. What the equivalences mean is that given suitable notions of reduction, as in the proposition, a proof of a formula A may itself act as a realizer of A.

1.5 Models

Intuitionistic logic has a standard truth-value semantics in Heyting algebras.

$$\frac{\overset{D}{\Gamma \rhd N : A} \qquad \dfrac{}{\Gamma, x : A \rhd x : A}}{\Gamma \rhd x[x := N] : A}\,\text{Cut}_x \qquad = \qquad \overset{D}{\Gamma \rhd N : A}$$

$$\frac{\overset{D}{\Gamma \rhd N : A} \qquad \dfrac{}{\Gamma, y : A \rhd y : A}}{\Gamma \rhd y[x := N] : A}\,\text{Cut}_x \qquad = \qquad \frac{}{\Gamma, y : A \rhd y : A}$$

$$\frac{\overset{D}{\Gamma \rhd N : A} \qquad \dfrac{\overset{E}{\Gamma, x : A \rhd M : B \to C} \qquad \overset{E'}{\Gamma, x : A \rhd M' : B}}{\Gamma, x : A \rhd MM' : C}}{\Gamma \rhd (MM')[x := N] : C}\,\text{Cut}_x$$

$$=$$

$$\frac{\dfrac{\overset{D}{\Gamma \rhd N : A} \qquad \overset{E}{\Gamma, x : A \rhd M : B \to C}}{\Gamma \rhd M[x := N] : B \to C} \qquad \dfrac{\overset{D}{\Gamma \rhd N : A} \qquad \overset{E'}{\Gamma, x : A \rhd M' : B}}{\Gamma \rhd M'[x := N] : B}}{\Gamma \rhd (M[x := N]\ M'[x := N]) : C}$$

$$\frac{\overset{D}{\Gamma \rhd N : A} \qquad \dfrac{\overset{E}{\Gamma, x : A, y : B \rhd M : C}}{\Gamma, x : A \rhd \lambda y.M : B \to C}}{\Gamma \rhd (\lambda y.M)[x := N] : B \to C}\,\text{Cut}_x$$

$$=$$

$$\frac{\dfrac{\overset{D}{\Gamma \rhd N : A} \qquad \overset{E}{\Gamma, x : A, y' : B \rhd M[y := y'] : C}}{\Gamma, y' : B \rhd M[x := N] : C}\,\text{Cut}_x}{\Gamma \rhd \lambda y'.M[y := y'][x := N] : B \to C}$$

Figure 1.4.1: Substitution in $\lambda^{\to}$

1.5.1 Definition. A *lattice* is a poset in which every pair of elements a, b has both a least upper bound, called $\sup\{a, b\}$ or join $a \vee b$, and a greatest lower bound, called $\inf\{a, b\}$ or meet $a \wedge b$. A lattice is *bounded* if it has least and greatest elements 0 and 1 (which are the units of $\vee$ and $\wedge$ respectively). Because $\vee$ and $\wedge$ are easily shown to be associative, a bounded lattice is therefore a poset that has all finite (including empty) joins and meets.

1.5.2 Definition. A *Heyting algebra* is a bounded lattice H where for all $a \in H$ the function $x \mapsto x \wedge a$ has an upper adjoint (def. 1.1.4) $y \mapsto a \Rightarrow y$. The operation $a \Rightarrow b$ thus denotes the greatest c such that $a \wedge c \leq b$. This means that $a \leq b$ iff $a \Rightarrow b = 1$. Negation $\neg a$ is defined as $a \Rightarrow 0$.

1.5.3 Definition. A *valuation* on a Heyting algebra H is a function $v : \mathcal{P} \to H$ from proposition letters to elements of H. Given a formula A and a valuation v, the *denotation* $[\![A]\!]_v$ of A with respect to v is defined by induction as follows, where $P \in \mathcal{P}$ and Γ is a sequence of formulas:

$$\begin{array}{rcl}
[\![P]\!]_v & = & v(P) \\
[\![\bot]\!]_v & = & 0 \\
[\![A \to B]\!]_v & = & [\![A]\!]_v \Rightarrow [\![B]\!]_v \\
[\![A \vee B]\!]_v & = & [\![A]\!]_v \vee [\![B]\!]_v \\
[\![A \wedge B]\!]_v & = & [\![A]\!]_v \wedge [\![B]\!]_v \\
\\
[\![\Gamma]\!]_v & = & [\![\bigwedge \Gamma]\!]_v
\end{array}$$

If $[\![A]\!]_v = 1$, we say that v *satisfies* A, written $v \vDash A$. A is *valid* if $v \vDash A$ for all valuations v, written simply $\vDash A$. If, for all v, $v \vDash A$ implies $v \vDash B$ then we write $A \vDash B$, and similarly for sequences of formulas Γ.

We state the following standard result without proof.

1.5.4 Proposition.

- *NJ is* sound*:* $\Gamma \vdash_{NJ} A$ *implies* $\Gamma \vDash A$
- *NJ is* complete*:* $\Gamma \vDash A$ *implies* $\Gamma \vdash_{NJ} A$.

A lattice, being a preorder, can only model the entailment relation, that is, the existence or otherwise of a proof of $A \vdash B$. To distinguish between different proofs of the same entailment, we need to move to categories, where hom sets may have more than one element.

1.5.5 Definition. A *cartesian* category $\mathbf{C}$ is a category with all finite products — equivalently, one with binary products and a terminal object. That is, for all $A, B, C \in \mathbf{C}$, the maps π_1, π_2 exist, and for any $f : C \to A, g : C \to B$, the dotted arrows exist uniquely such that the diagram on the right commutes.

$$\begin{tikzcd}
A \arrow[dd, dotted, "\diamond_A" description] & & & C \arrow[ddl, "f"'] \arrow[dd, dotted, "\langle f, g \rangle" description] \arrow[ddr, "g"] & \\
& & & & \\
\top & & A & A \times B \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] & B
\end{tikzcd}$$

The definition of a *cocartesian* category is dual — it is a category with all finite coproducts, i.e. binary coproducts $A+B$ and an initial object $\bot$, with mediating morphisms written as $[f,g]$ and $\Box_A$, for all A, B, C and $f : A \to C$, $g : B \to C$.

1.5.6 Remark. Observe that in any cartesian category, by uniqueness of $\Diamond$, we have $\Diamond_\top = 1_\top$ and, for all $f : A \to B$, $\Diamond_B \circ f = \Diamond_A$; i.e. the following diagram commutes:

$$\begin{tikzcd}
A \arrow[dd, "f"'] \arrow[dr, "\Diamond_A"] & \\
& \top \\
B \arrow[ur, "\Diamond_B"'] &
\end{tikzcd}$$

The dual holds in any cocartesian category: $\Box_\bot = 1_\bot$ and $f \circ \Box_A = \Box_B$.

1.5.7 Definition. A *cartesian closed* category (CCC) is a cartesian category $\mathbf{C}$ where for each object $A \in \mathbf{C}$ the functor $- \times A$ has a right adjoint $A \Rightarrow -$. The adjunction $\langle - \times A, A \Rightarrow -, \phi^A \rangle$, where $\phi^A : \mathbf{C}(- \times A, -) \cong \mathbf{C}(-, A \Rightarrow -)$ is the isomorphism of adjunction, gives rise to the following two-way inference rule:

$$\begin{array}{c} f = \tilde{\phi}^A(g) : B \times A \to C \\ \hline\hline g = \phi^A(f) : \qquad B \to A \Rightarrow C \end{array}$$

and to the usual natural transformations (unit and counit):

$$\begin{array}{lcllcl} \eta^A & : & 1_{\mathbf{C}} \stackrel{\cdot}{\longrightarrow} A \Rightarrow (- \times A) & \eta^A_B & = & \phi^A(1_{B \times A}) \\ \epsilon^A & : & (A \Rightarrow -) \times A \stackrel{\cdot}{\longrightarrow} 1_{\mathbf{C}} & \epsilon^A_C & = & \tilde{\phi}^A(1_{A \Rightarrow C}) \end{array}$$

A *bicartesian closed category* (BCC) is a CCC with all finite coproducts. An *almost bicartesian closed category* (ACC) is a CCC with n-ary coproducts for $n \geq 2$ — that is, a BCC that need not have an initial object.

To interpret a proof of NM (or a term of $\lambda^{\to\times+}$) in an ACC $\mathbf{C}$, we simply generalize def. 1.5.3. That is, the denotation of a derivation

$$D :: x_1 : A_1, \ldots, x_n : A_n \ \triangleright\ B$$

instead of being an inequality

$$\bigwedge_i [\![A_i]\!] \leq [\![B]\!]$$

will be an arrow in $\mathbf{C}$:

$$\prod_i [\![A_i]\!] \xrightarrow{[\![D]\!]} [\![B]\!]$$

1.5.8 Definition. An interpretation of a typed λ-calculus L in a BCC $\mathbf{C}$ is given by the assignment of an object of $\mathbf{C}$ to each atomic type of L. The interpretation is extended to compound types by

$$\begin{array}{rcl} [\![\bot]\!] & = & \bot \\ [\![A \times B]\!] & = & [\![A]\!] \times [\![B]\!] \\ [\![A \to B]\!] & = & [\![A]\!] \Rightarrow [\![B]\!] \\ [\![A + B]\!] & = & [\![A]\!] + [\![B]\!] \end{array}$$

$$[\![x_1 : A_1, \ldots, x_n : A_n]\!] \quad = \quad [\![A_1]\!] \times \cdots \times [\![A_n]\!]$$

Typing derivations (and thus terms, cf. proposition 1.4.13) are interpreted as follows, where $\phi, \tilde{\phi}$ are as in def. 1.5.7, and $\delta_A : A \to A \times A$ is the diagonal map.

$$\begin{aligned}
\llbracket \Gamma \rhd M[x := N] : B \rrbracket &= \llbracket \Gamma, x : A \rhd M : B \rrbracket \\
&\quad \circ (\Gamma \times \llbracket \Gamma \rhd N : A \rrbracket) \circ \delta_\Gamma \\[1ex]
\llbracket x_1 : A_1, \ldots, x_n : A_n \rhd x_i : A_i \rrbracket &= \pi_i \\[1ex]
\llbracket \Gamma \rhd \langle M, N \rangle : A \times B \rrbracket &= \langle \llbracket \Gamma \rhd M : A \rrbracket, \llbracket \Gamma \rhd N : B \rrbracket \rangle \\
\llbracket \Gamma \rhd \pi_i M : A_i \rrbracket &= \pi_i \circ \llbracket \Gamma \rhd M : A_1 \times A_2 \rrbracket \\[1ex]
\llbracket \Gamma \rhd \iota_i M : A_1 + A_2 \rrbracket &= \iota_i \llbracket \Gamma \rhd M : A_i \rrbracket \\
\llbracket \Gamma \rhd [x.N_1 | y.N_2] M : C \rrbracket &= [\llbracket \Gamma, x : A \rhd N_1 : C \rrbracket, \llbracket \Gamma, y : B \rhd N_2 : C \rrbracket] \\
&\quad \circ \llbracket \Gamma \rhd M : A + B \rrbracket \\[1ex]
\llbracket \Gamma \rhd \lambda x.M : A \to B \rrbracket &= \phi^A_{\Gamma,B}(\llbracket \Gamma, x : A \rhd M : B \rrbracket) \\
\llbracket \Gamma, x : A \rhd M^{A \to B} x : B \rrbracket &= \tilde{\phi}^A_{\Gamma,B}(\llbracket \Gamma \rhd M : A \to B \rrbracket)
\end{aligned}$$

The last clause is extended to cover all applications by observing that $MN = (Mx)[x := N]$ for $x \notin \mathrm{FV}(M)$.

This interpretation is sound with respect to $=_{\beta\eta}$ because by def. 1.5.7

$$\begin{aligned}
\llbracket \Gamma \rhd \lambda x.M \; N^A : B \rrbracket &= \tilde{\phi}^A_{\Gamma,B} \phi^A_{\Gamma,B}(\llbracket \Gamma, x : A \rhd M : B \rrbracket) \\
&\quad \circ (\Gamma \times \llbracket \Gamma \rhd N : B \rrbracket) \circ \delta_\Gamma \\
&= \llbracket \Gamma, x : A \rhd M : B \rrbracket \circ (\Gamma \times \llbracket \Gamma \rhd N : A \rrbracket) \circ \delta_\Gamma \\
&= \llbracket \Gamma \rhd M[x := N] : B \rrbracket
\end{aligned}$$

and similarly for $=_\eta$ and the other type constructors.

Because this interpretation is sound, any ACC is a model of NM and of $\lambda^{\to}$, for some choice of atomic propositions or types.

1.5.9 Remark. Until now, we have not discussed intuitionistic logic, with its extra rule of *ex falso quodlibet* $\bot \vdash A$ for all A. We can now interpret this rule in any BCC by means of the initial object $\bot$. This amounts to adding a constant $\mathcal{A}$ to $\lambda^{\to}$, with the following rules (see [Gal93]):

$$\frac{\Gamma \rhd M : \bot}{\Gamma \rhd \mathcal{A}_A(M) : A}$$

$$\begin{aligned}
(\mathcal{A}_{A \to B}(M) \; N) &\succ \mathcal{A}_B(M) \\
\pi_i \mathcal{A}_{A_1 \times A_2}(M) &\succ \mathcal{A}_{A_i}(M) \\
[x.N^C_1 | y.N^C_2] \mathcal{A}_{A+B}(M) &\succ \mathcal{A}_C(M)
\end{aligned}$$

The $\mathcal{A}$ combinator has the effect of discarding terms surrounding its argument. It can thus be thought of as an *exit* operator.

Although there is no problem with interpreting intuitionistic logic in bicartesian closed categories, attempting to extend the correspondence to classical logic in the obvious way — by making the canonical arrow $\llbracket \lambda xy.yx \rrbracket : A \to \neg\neg A$ into an isomorphism — fails dramatically, as a result of the following.

1.5.10 Proposition (Joyal)**.** *In a CCC with initial object, if* $\hom(A, \bot)$ *is nonempty then* $A \cong \bot$*.*

Proof. ([LS86, Prop. 8.3, p. 67]) Given an arrow $g : A \to \bot$, the following diagram commutes — the left square by definition of π_2 (def. 1.5.5), the right by remark 1.5.6:

$$\begin{tikzcd}
A \arrow[r, "g"] \arrow[d, "\langle 1_A, g \rangle"'] & \bot \arrow[r, "\Box_A"] \arrow[d, equal] & A \\
A \times \bot \arrow[r, "\pi_2"'] & \bot \arrow[r, "\Box_{A \times \bot}"'] & A \times \bot \arrow[u, "\pi_1"']
\end{tikzcd}$$

Since $\hom(A \times \bot, A \times \bot) \cong \hom(\bot, A \Rightarrow A \times \bot) = \{\Box_{A \Rightarrow A \times \bot}\}$ is a one-element set, the bottom composite $\Box_{A \times \bot}\pi_2 = 1_{A \times \bot}$. Thus

$$\begin{array}{rcl}
\Box_A \circ g & = & \pi_1 \circ \Box_{A \times \bot} \circ \pi_2 \circ \langle 1_A, g \rangle \\
& = & \pi_1 \circ \langle 1_A, g \rangle \\
& = & 1_A
\end{array}$$

and

$$g \circ \Box_A \ = \ \Box_\bot \ = \ 1_\bot$$

so $A \cong \bot$. In particular, $\hom(A, \bot)$ contains at most one element. □

Corollary. *A 'Boolean category' — a CCC* **B** *with initial object where $A \cong \neg\neg A$ for all A — is equivalent to a Boolean algebra.*

Proof. Suppose $A \cong \neg\neg A$ for all A. Then for any objects A and B,

$$\begin{array}{rcll}
\hom(B, A) & \cong & \hom(B, \neg\neg A) & \text{(Yoneda)} \\
& \cong & \hom(B \times \neg A, \bot) & (- \times \neg A \dashv \neg A \Rightarrow -) \\
& \cong & \{*\} \text{ or } \varnothing & \text{(Prop. 1.5.10)}
\end{array}$$

Thus **B** is a preorder. The equivalence **BoolCat** $\simeq$ **BoolAlg** is easily checked. □

This shows that the notion of a model of classical proofs is not as straightforward as it may seem. Indeed, we will see in section 2.5 that the correct such notion is still being sought, while remark 2.3.6 shows how this issue relates to the question of giving an appropriate notion of equality for classical proofs.

We have now covered, though only superficially, all three aspects of the modern approach to the Curry–Howard correspondence for minimal logic — logic, λ-calculus and category theory. The correspondence can be extended to predicate logic and type theory: see [LS86] for details. Here, however, we will examine a similar correspondence for sequent calculus, and show how classical logic may be given a natural computational interpretation.

2 Classical Logic

From what we have seen, it would seem counter-intuitive that classical logic could have any computational interpretation. Nevertheless, we will see that, by expressing the Curry–Howard correspondence in terms of sequent calculus

$$\dfrac{M : C_1 \to C_2 \to \cdots \to C_m \to B \qquad N_1 : C_1}{MN_1 : C_2 \to \cdots \to C_m \to B}$$

$$\vdots$$

$$\dfrac{\dfrac{MN_1N_2 \cdots N_{m-1} : C_m \to B \qquad\qquad N_m : C_m}{MN_1N_2 \cdots N_{m-1}N_m : B}}{\lambda x_n.MN_1N_2 \cdots N_m : A_n \to B}$$

$$\vdots$$

$$\dfrac{\lambda x_2 \cdots \lambda x_n.MN_1N_2 \cdots N_m : A_2 \to \cdots \to A_n \to B}{\lambda x_1.\lambda x_2 \cdots \lambda x_n.MN_1N_2 \cdots N_m : A_1 \to A_2 \to \cdots \to A_n \to B}$$

Figure 2.1.1: Form of $\lambda^{\to}$ derivations

rather than natural deduction, not only do we gain an insight into the structure of proofs and λ-terms, but we also find a natural way of interpreting classical proofs.

We begin this section with an informal motivation of sequent calculus, both as a proof-system and as a type-system.

2.1 The Structure of λ-Terms

Many properties of λ-terms follow from this simple but important fact:

2.1.1 Proposition. *Any λ-term is of the following form $(*)$, for some $n, m \in \mathbb{N}$:*

$$\lambda x_1.\lambda x_2 \cdots \lambda x_n.MN_1N_2 \cdots N_m \qquad (*)$$

where M is either an abstraction or a variable, and each N_i is also of this form.

Proof. Let $(*)^a_b$ denote the predicate 'of the form $(*)$, with $n = a, m = b$'. By induction on terms, we have:

Case 1. A variable x is $(*)^0_0$.

Case 2. An abstraction $\lambda x.N$, where N is $(*)^a_b$, is $(*)^{a+1}_b$.

Case 3. An application NN', where N is $(*)^a_b$, is $(*)^0_{b+1}$.

That M above is not an application follows from the finiteness of terms (and, in particular, that of b in case 3). □

2.1.2 Remark. It follows from the above that any $\lambda^{\to}$ derivation is of the form shown in figure 2.1.1, where M is either a variable or an abstraction (type environments are elided). That is, a derivation consists of a sequence of eliminations followed by a sequence of introductions. It can be reconstructed by starting with the switchover point (here the judgement $MN_1N_2 \cdots N_{m-1}N_m : B$) and using introduction rules to grow the tree downwards (i.e. at the root, as usual) and elimination rules to grow it upwards (i.e. by operating on hypotheses).

The upward $\to^-$ rule will take derivations $\Gamma \vartriangleright M : A$ and $\Gamma, x : B \vartriangleright N : C$ and produce one of $\Gamma, y : A \to B \vartriangleright N[x := yM] : C$.

$$\begin{array}{c}\Gamma\\ \vdots\\ M:A\end{array} \quad , \quad \begin{array}{c}\Gamma, x:B\\ \vdots\\ N:C\end{array} \quad \mapsto \quad \begin{array}{c} \qquad\qquad \Gamma \\ \qquad\qquad \vdots \\ \dfrac{y:A\to B \qquad M:A}{(y\ M):B}\to^- \\ \vdots \\ N[x:=(y\ M)]:C \end{array}$$

This suggests the following typing rule:

$$\dfrac{\Gamma \triangleright M:A \qquad \Gamma, x:B \triangleright N:C}{\Gamma, y:A\to B \triangleright N[x:=(y\ M)]:C}\to_L^\lambda$$

2.1.3 Definition. The implicative minimal sequent calculus LM$^\to$ is defined by the following rules:

$$\dfrac{}{\Gamma, A \triangleright A}\text{Ax}$$

$$\dfrac{\Gamma \triangleright B}{\Gamma, A \triangleright B}\text{Wk}_L \qquad\qquad \dfrac{\Gamma, A, A \triangleright B}{\Gamma, A \triangleright B}\text{Cn}_L$$

$$\dfrac{\Gamma \triangleright A \qquad \Gamma, A \triangleright B}{\Gamma \triangleright B}\text{Cut}$$

$$\dfrac{\Gamma \triangleright A \qquad \Gamma, B \triangleright C}{\Gamma, A\to B \triangleright C}\to_L \qquad \dfrac{\Gamma, A \triangleright B}{\Gamma \triangleright A\to B}\to_R$$

The intuitionistic calculus LJ$^\to$ is the same, but extended to negation with the following rule:

$$\dfrac{}{\bot \triangleright}\bot_L$$

It is instructive to think of an LM derivation as building an NM derivation from the inside out, as in the remark. The axiom rule denotes the singleton deduction $D :: A \vdash A$. Left contraction Cn_L allows two hypothesis-packets to be merged into one, and left weakening Wk_L allows the formation of empty packets. The $\to_R$ rule is introduction as before, and the $\to_L$ rule constructs a derivation with a hypothesis $A \to B$ from two others, as in the remark. Finally, Cut allows substitution of derivations for hypotheses, just as in figure 1.4.1.

2.1.4 Remark. Removing the 'structural' rules of Cn and Wk yields (multiplicative) *intuitionistic linear logic* MILL[5], so called because a linear sequent $A \triangleright B$ expresses the fact that B can be derived using A exactly once. The structural rules thus correspond to duplication (Cn) and discarding (Wk) of hypotheses, or variables.

[5] Strictly speaking, the binary rules of MILL have different contexts Γ, Δ in their premises, rather than the same Γ in both. The literature on linear logic is vast — some good references are [Gir87, Wad93, Abr93, BS04].

2.2 Interpreting Sequent Calculus

2.2.1 Remark. Herbelin observes in [Her94] that the term notation for LM$^{\to}$ suggested by the rule $\to_L^\lambda$ (remark 2.1.2) can assign the same term to distinct proofs of the same sequent:

$$\dfrac{\dfrac{\dfrac{}{x:A,w:C \rhd x:A}\,\text{Ax} \qquad \dfrac{}{x:A,w:C,y:B \rhd y:B}\,\text{Ax}}{z:A\to B, x:A, w:C \rhd zx:B}\,\to_L}{z:A\to B, x:A \rhd \lambda w.zx : C\to B}\,\to_R$$

$$\dfrac{\dfrac{}{x:A \rhd x:A}\,\text{Ax} \qquad \dfrac{\dfrac{}{x:A,w:C,y:B \rhd y:B}\,\text{Ax}}{x:A,y:B \rhd \lambda w.y : C\to B}\,\to_R}{z:A\to B, x:A \rhd \lambda w.zx : C\to B}\,\to_L$$

To get around this, we observe that, in the term $N[x := (y\ M)]$, the fresh variable y introduced by $\to_L^\lambda$ must occur *linearly* (i.e. exactly once) as long as x occurs linearly, which it will in an axiom sequent. In fact, these variables serve only as placeholders that mark where the next argument (say M') will be inserted. For example, when the rule is applied twice in succession, we get:

$$\dfrac{\Gamma \rhd M' : A \qquad \dfrac{\Gamma \rhd M : B \qquad \Gamma, x:C \rhd N : D}{\Gamma, y : B\to C \rhd N[x := (y\ M)] : D}\,\to_L^\lambda}{\Gamma, z : A\to B\to C \rhd N[x := ((z\ M')\ M)] : D}$$

where $N[x := ((z\ M')\ M)]$ is short for $N[x := (y\ M)][y := (z\ M')]$. We choose a new constant to represent this placeholder, and formulate typing rules that enforce its linearity. This has the effect of disallowing substitution under λ, and so the second derivation above becomes invalid.

2.2.2 Definition. λ-*contexts* K are defined by the following grammar, where M ranges over Λ and $[\,\cdot\,]$ (*hole*) is a new constant:

$$K \quad ::= \quad [\,\cdot\,] \mid (K\ M)$$

Substitution $[M]K$ of a term for $[\,\cdot\,]$ is defined thus:

$$\begin{array}{rcl} [M][\,\cdot\,] & = & M \\ [M](K\ N) & = & ([M]K\ N) \end{array}$$

Given contexts K and J, the context $[J]K$ is defined similarly.

A context is thought of as a term with a unique 'hole', represented by $[\,\cdot\,]$. The notation $[M]K$ represents the term got by plugging M into the hole of K. The hole of a compound context $[J]K$ is the hole of J.

2.2.3 Definition ($\bar{\lambda}$-calculus [Her94])**.** Terms of the $\bar{\lambda}$-calculus (in fact, a syntactic variant of it) are defined as follows, where M is a λ-term and K a λ-context. Substitutions are now part of the object language and are denoted $M\{x := N\}$:

$$\begin{array}{rcl} M & ::= & x \mid \lambda x.M \mid [M]K \mid M\{x := M\} \\ K & ::= & [\,\cdot\,] \mid ([\,\cdot\,]\ M) \mid [K]K \mid K\{x := M\} \end{array}$$

Observe that the first three productions for K are equivalent to the two given in def. 2.2.2. Reduction rules are as follows:

$$\begin{array}{rcl}
[(([\lambda x.M]\ N))]K & \succ & [M\{x := N\}]K \\
\\
[M] & \succ & M \\
[[M]K]K' & \succ & [M][K]K' \\
\\
x\{x := M\} & \succ & M \\
y\{x := M\} & \succ & y \\
(\lambda z.N)\{x := M\} & \succ & \lambda z.(N\{x := M\}) \\
\\
[\,\cdot\,]\{x := M\} & \succ & [\,\cdot\,] \\
([(([\,\cdot\,]\ N))]K)\{x := M\} & \succ & [(([\,\cdot\,]\ N\{x := M\}))]K\{x := M\},
\end{array}$$

with the usual renaming of bound variables to avoid capture in the substitution case for λ.

2.2.4 Remark. λ-contexts are effectively *lists* of terms, with a hole for a head subterm (see def. 2.2.7):

$$K = (\ldots(([\,\cdot\,]\ N_1)\ N_2)\ldots\ N_m)$$

They can be generated from $[\,\cdot\,]$ by successively substituting the 'singleton' context $([\,\cdot\,]\ N_i)$ for $[\,\cdot\,]$. Comparing this with remark 2.1.2, we can derive the typing rule

$$\frac{\Gamma\ \rhd\ M : A \qquad \Gamma, [\,\cdot\,] : B\ \rhd\ K : C}{\Gamma, [\,\cdot\,] : A \to B\ \rhd\ [(([\,\cdot\,]\ M))]K : C}$$

This says that from a term of type A and a context of type C with a hole of type B we can construct a context (still of type C) with a hole of type $A \to B$ in the indicated way. Substituting a term for $[\,\cdot\,]$ in this new context will apply the term to $M : A$ and send the result (of type B) to K — hence the type $A \to B$.

Terms derived using this rule will not in general be unique, as in remark 2.2.1. Since left contraction duplicates variables in terms (see remark 2.1.4), we enforce linearity of $[\,\cdot\,]$ by restricting contraction on the type of $[\,\cdot\,]$ in the following definition.

2.2.5 Definition (LJT [Her94])**.** The system LJT is defined in figure 2.2.1, adapted from [Her94] for our variant of $\bar{\lambda}$. Sequents are of the form $\Gamma\ \rhd\ M : A$ for terms M and type environments Γ, or $\Gamma\,|\,[\,\cdot\,] : A\ \rhd\ K : B$ for contexts K, where A is the type of K's hole. The place delimited by $|$ and $\rhd$ is called the *stoup*, and contains exactly one typing $[\,\cdot\,] : A$ for some type A. Sequents $\Gamma\ \rhd\ M : A$ can be thought of as having an empty stoup.

The Cut rules $C_{H,M}$ have their labels set above to save horizontal space. The rules in the left column type contexts while those in the right type terms. The rules C_H and C_M are called *head cut* and *mid cut* respectively, and are used according to whether the cut-formula is in the stoup of the right-hand premise or not.

Observe that in the context-typing rules the output type (B or C) of the context K or L never changes, so what is being typed is in fact the hole of the context.

$$\frac{}{\Gamma\,|\,[\,\cdot\,]:A \rhd [\,\cdot\,]:A}\;\text{Ax} \qquad\qquad \frac{\Gamma, x:A\,|\,[\,\cdot\,]:A \rhd K:B}{\Gamma, x:A \rhd [x]K:B}\;\text{Cn}$$

$$\frac{\Gamma \rhd M:A \qquad \Gamma\,|\,[\,\cdot\,]:B \rhd K:C}{\Gamma\,|\,[\,\cdot\,]:A \to B \rhd [([\,\cdot\,]\ M)]K:C}\;\to_L \qquad\qquad \frac{\Gamma, x:A \rhd M:B}{\Gamma \rhd \lambda x.M:A \to B}\;\to_R$$

$$\begin{array}{cc} C_H\text{:} & C_H\text{:} \\ \dfrac{\Gamma\,|\,[\,\cdot\,]:C \rhd K:A \qquad \Gamma\,|\,[\,\cdot\,]:A \rhd L:B}{\Gamma\,|\,[\,\cdot\,]:C \rhd [K]L:B} & \dfrac{\Gamma \rhd M:A \qquad \Gamma\,|\,[\,\cdot\,]:A \rhd K:B}{\Gamma \rhd [M]K:B} \end{array}$$

$$\begin{array}{cc} C_M\text{:} & C_M\text{:} \\ \dfrac{\Gamma \rhd M:A \qquad \Gamma, x:A\,|\,[\,\cdot\,]:C \rhd K:B}{\Gamma\,|\,[\,\cdot\,]:C \rhd K\{x:=M\}:B} & \dfrac{\Gamma \rhd M:A \qquad \Gamma, x:A \rhd N:B}{\Gamma \rhd N\{x:=M\}:B} \end{array}$$

Figure 2.2.1: LJT

$$\cfrac{\cfrac{\cfrac{\cfrac{\Gamma \rhd M:C \qquad \cfrac{\Gamma \rhd N_1:C_1 \qquad \cfrac{\begin{array}{c}\cfrac{\Gamma \rhd N_m:C_m \qquad |\,[\,\cdot\,]:B \rhd [\,\cdot\,]:B}{\Gamma\,|\,[\,\cdot\,]:C_m \to B \rhd [\,\cdot\,]N_m:B}\\ \vdots\end{array}}{\Gamma\,|\,[\,\cdot\,]:C_2 \to \cdots \to C_m \to B \rhd [\,\cdot\,]N_2 \cdots N_m:B}}{\Gamma\,|\,[\,\cdot\,]:C \rhd [\,\cdot\,]N_1N_2 \cdots N_m:B}}{\Gamma \rhd MN_1 \cdots N_m:B}\;\text{C}_H}{\Gamma \setminus \{x_n\} \rhd \lambda x_n.MN_1 \cdots N_m:A_n \to B}}{\begin{array}{c}\vdots\\ \Gamma \setminus \{x_2,\dots,x_n\} \rhd \lambda x_2 \dots x_n.MN_1 \cdots N_m:A_2 \to \cdots A_n \to B\end{array}}}{\Gamma \setminus \{x_1,\dots,x_n\} \rhd \lambda x_1 \dots x_n.MN_1 \cdots N_m:A_1 \to \cdots A_n \to B}$$

Figure 2.2.2: Form of LJT derivations

2.2.6 Remark. By analogy with remark 2.1.2, any LJT derivation has the form shown in figure 2.2.2 (see [Her94]), where $C = C_1 \to \cdots \to C_m \to B$.

The Structure of Contexts

We now have a λ-calculus that is a Curry–Howard-style term calculus for LM, although it is still far from obvious how to extend $\bar{\lambda}$ to deal with classical logic. A more elegant variant of $\bar{\lambda}$ is $\bar{\lambda}\bar{\mu}$, defined below. We will see that the natural extension to classical logic of $\bar{\lambda}\bar{\mu}$ in effect allows arbitrary context variables in place of $[\,\cdot\,]$.

2.2.7 Definition. In any λ-term $\lambda x_1.\lambda x_2 \cdots \lambda x_n.MN_1N_2 \cdots N_m$ as in proposition 2.1.1, if M is an abstraction $\lambda y.M'$, then the subterm $(\lambda y.M'\ N_1)$ is called the *head redex* of the term, and if M is a variable (the *head variable*) then the term is in *head normal form* (HNF). If a term is in HNF or $n > 0$ then it is in *weak head normal form* (WHNF). For convenience, we will refer to head variables and operators of head redexes as *head subterms*.

2.2.8 Definition. *Head reduction* is the reduction strategy that only reduces the head redex of a term. *Weak head reduction* is the same, except it never reduces a redex that is inside the scope of a λ.

Clearly, if (weak) head reduction terminates, then the result is a (weak) head normal form. Weak head reduction corresponds to the strategy usually found in functional programming languages, where the bodies of functions are not evaluated until all arguments have been supplied.

2.2.9 Definition. The Krivine abstract machine (K-machine) performs weak head reduction using stacks E of λ-terms, where e (a constant) is the empty stack:

$$E ::= e \mid M \cdot E$$

States of the machine are pairs $\langle M \| E\rangle$. Its transitions are:

$$\begin{array}{rcl} \langle (M\ N) \| E\rangle & \succ & \langle M \| N \cdot E\rangle \\ \langle \lambda x.M \| N \cdot E\rangle & \succ & \langle M[x := N] \| E\rangle \end{array}$$

A term M is reduced by starting the machine with $\langle M \| e\rangle$. An alternative is to replace e with a set $\{\alpha, \beta, \ldots\}$ of *covariables* to range over stacks, and start the machine with $\langle M \| \alpha\rangle$, where α is intended to represent some unspecified outer context.

2.2.10 Remark. Given any term L, the K-machine will, by means of the first transition only, reach a state

$$\langle M \| N_1 \cdot N_2 \cdots N_n \cdot e\rangle,$$

where M is the head subterm of L. Clearly, L is in WHNF if and only if L is a λ-abstraction (and so $M = L$ and $n = 0$) or $M = x$ for some variable x. The stack $N_1 \cdot N_2 \cdots N_n \cdot e$ is known in the functional programming literature as the *spine stack* of L.

The above state $\langle M \| N_1 \cdot N_2 \cdots N_n \cdot e\rangle$ is derived straightforwardly from the term $M N_1 N_2 \cdots N_n$. This suggests that we may view it as representing the $\bar\lambda$-term $[M]K$, where $K = [\,\cdot\,]N_1 N_2 \cdots N_n$, as in remark 2.2.4. In particular, the spine stack represents the context K.

2.2.11 Definition. (Adapted from [Her05], sec. 2.13) The $\bar\lambda\bar\mu$-calculus consists of terms M, *coterms* E and *commands* (K-machine states) P:

$$\begin{array}{rcl} P & ::= & \langle M \| E\rangle \\ M & ::= & x \mid \lambda x.M \mid \widehat{P} \\ E & ::= & e \mid M \cdot E \mid \bar\mu x.P \end{array}$$

Reductions are as follows:

$$\begin{array}{rcl} \langle \lambda x.M \| N \cdot E\rangle & \succ & \langle M[x := N] \| E\rangle \\ \langle \widehat{\langle M \| E\rangle} \| E'\rangle & \succ & \langle M \| E[e := E']\rangle \\ \langle M \| \bar\mu x.P\rangle & \succ & P[x := M] \end{array}$$

The substitution $E[e := E']$ is defined in the obvious way as

$$\begin{array}{rcl}(N_1 \cdots N_n \cdot e)[e := E'] & = & N_1 \cdots N_n \cdot E' \\ (N_1 \cdots N_n \cdot \bar\mu x.\langle M \| E''\rangle)[e := E'] & = & N_1 \cdots N_n \cdot \bar\mu x.\langle M \| E''[e := E']\rangle\end{array}$$

with the usual capture-avoiding renaming of x in the second rule.

The constructor $\hat{\cdot}$ makes a term out of a command by binding e, while $\bar\mu x$ turns a command into a coterm by singling out an input. Intuitively, commands are thought of as *processes*, with no specified input or output channels, while a term is a process with a specified *continuation* or output channel, and a coterm is a continuation, i.e. a process with a specified input channel. The restriction to a single covariable e forces terms to be linear in their continuations. We will see that to extend $\bar\lambda\bar\mu$ to classical logic we need only remove this restriction.

As observed in def. 2.2.5, the context rules of LJT only manipulate the type of a context's hole, not its output type. We may thus reformulate them in terms of spine stacks as follows. Note that now stacks appear inside the stoup, and the judgement $E : A$ means 'the hole of E is of type A'.

2.2.12 Definition. The variant LJT$_{\bar\mu}$ of LJT is defined by the following rules.

$$\frac{}{\Gamma, x : A \;\rhd\; x : A}\,\text{Ax} \qquad\qquad \frac{}{\Gamma \,|\, e : A \;\rhd\; e : A}\,\text{Ax}$$

$$\frac{\Gamma, x : A \;\rhd\; M : B}{\Gamma \;\rhd\; \lambda x.M : A \to B}\,\to_R \qquad \frac{\Gamma \;\rhd\; M : A \qquad \Gamma \,|\, E : B \;\rhd\; e : C}{\Gamma \,|\, M \cdot E : A \to B \;\rhd\; e : C}\,\to_L$$

$$\frac{\Gamma \;\rhd\; M : A \qquad \Gamma \,|\, E : A \;\rhd\; e : B}{\langle M \| E\rangle : (\Gamma \;\rhd\; e : B)}\,\text{Cut}$$

$$\frac{P : (\Gamma \;\rhd\; e : A)}{\Gamma \;\rhd\; \hat{P} : A}\,\mu \qquad\qquad \frac{P : (\Gamma, x : A \;\rhd\; e : B)}{\Gamma \,|\, \bar\mu x.P : A \;\rhd\; e : B}\,\bar\mu$$

It is not obvious that the system above is confluent — there are two reduction rules that apply to the middle term below:

$$\langle M \| E[e := \bar\mu x.P]\rangle \preceq \langle \widehat{\langle M \| E\rangle} \| \bar\mu x.P\rangle \succeq P[x := \widehat{\langle M \| E\rangle}]$$

We define translations between $\bar\lambda\bar\mu$ and $\bar\lambda$ that preserve reduction and typing. Because the typeable terms of LJT are confluent (see [Her94]) so are those of $\bar\lambda\bar\mu$.

2.2.13 Definition. The translation $\overline{\cdot}$ from $\bar\lambda\bar\mu$ to $\bar\lambda$ is defined as follows:

$$\overline{\langle M \| E\rangle} \;=\; [\overline{M}]\overline{E}$$

$$\begin{array}{rclrcl}\overline{x} & = & x & \overline{e} & = & [\,\cdot\,] \\ \overline{\lambda x.M} & = & \lambda x.\overline{M} & \overline{N \cdot E} & = & [([\,\cdot\,]\ \overline{N})]\overline{E} \\ \overline{\hat{P}} & = & \overline{P} & \overline{\bar\mu x.P} & = & \overline{P}\{x := [\,\cdot\,]\}\end{array}$$

Observe that $\overline{E[e := E']} = [\overline{E}]\overline{E'}$.

The translation is extended to typing derivations in the obvious way. Note that the clause for $\bar\mu$ may yield a non-well-formed $\bar\lambda$ term, but this will not be a problem since we will only be applying the translation to commands $\langle M \| \bar\mu x.P\rangle$, yielding well-formed terms $\overline{P}\{x := \overline{M}\}$.

2.2.14 Definition. The translation $\underline{\;\cdot\;}$ from $\bar{\lambda}$ to $\bar{\lambda}\bar{\mu}$ is defined as follows:

$$\underline{[[M]K]K'} \;=\; \underline{[M][K]K'}$$

$$\begin{array}{rcl@{\qquad}rcl}
\underline{[M]K} & = & \langle \underline{M} \| \underline{K} \rangle & \underline{[K]K'} & = & \underline{K}[e := \underline{K'}] \\
\underline{x} & = & x & \underline{[\,\cdot\,]} & = & e \\
\underline{\lambda x.M} & = & \lambda x.\underline{M} & \underline{[[\,\cdot\,]\ M]K} & = & \underline{M} \cdot \underline{K} \\
\underline{M\{x := N\}} & = & \underline{M}[x := \underline{N}] & \underline{K\{x := M\}} & = & \underline{K}[x := \underline{M}]
\end{array}$$

The first clause ensures that in the clause for $[M]K$ the term M is not itself of the form $[M']K'$.

2.2.15 Proposition.

1. *If $P : (\Gamma \vdash_{\bar{\lambda}\bar{\mu}} e : A)$ then $\Gamma \vdash_{\bar{\lambda}} \overline{P} : A$.*
2. *If $\Gamma \vdash_{\bar{\lambda}} M : A$ then $\Gamma \vdash_{\bar{\lambda}\bar{\mu}} \underline{M} : A$, and if $\Gamma \,|\, [\,\cdot\,] : A \vdash_{\bar{\lambda}} K : B$ then $\Gamma \,|\, \underline{K} : A \vdash_{\bar{\lambda}\bar{\mu}} e : B$*
3. *If $P \succeq_{\bar{\lambda}\bar{\mu}} P'$ then $\overline{P} \succeq_{\bar{\lambda}} \overline{P'}$.*
4. *If $M \succeq_{\bar{\lambda}} M'$ then $\underline{M} \succeq_{\bar{\lambda}\bar{\mu}} \underline{M'}$.*
5. *For all $\bar{\lambda}\bar{\mu}$ commands P, $P \succeq \underline{\overline{P}}$.*

Proof. We prove (1), (3) and (5). The proofs of (2) and (4) are very similar to those of (1) and (3).

1. By straightforward induction on derivations. The case for $\bar{\mu}$ is:

$$\dfrac{\Gamma \rhd M : A \qquad \dfrac{P : (\Gamma, x : A \rhd e : B)}{\Gamma \,|\, \bar{\mu}x.P : A \rhd e : B}}{\langle M \| \bar{\mu}x.P \rangle : (\Gamma \rhd e : B)}$$

$$\mapsto$$

$$\dfrac{\Gamma \rhd \overline{M} : A \qquad \Gamma, x : A \rhd \overline{P} : B}{\Gamma \rhd \overline{P}\{x := \overline{M}\} : B}$$

3.

Case 1.

$$\langle \lambda x.M \| N \cdot E \rangle \succeq \langle M[x := N] \| E \rangle$$

The translated reduction holds by def. 2.2.3:

$$[([\lambda x.\overline{M}]\ \overline{N})]\overline{E} \;\succeq\; [\overline{M}\{x := \overline{N}\}]\overline{E} \;\succeq\; [\overline{M}[x := \overline{N}]]\overline{E}$$

Case 2.

$$\langle \widehat{\langle M \| E \rangle} \| E' \rangle \succeq \langle M \| E[e := E'] \rangle$$

By def. 2.2.13, $\overline{E[e := E']} = [\overline{E}]\overline{E'}$, so

$$[[\overline{M}]\overline{E}]\overline{E'} \succeq [\overline{M}][\overline{E}]\overline{E'}$$

holds by def. 2.2.3.

Case 3.
$$\langle M \| \bar{\mu}x.P \rangle \succeq P[x := M]$$

By the definition of substitution

$$\overline{P}\{x := \overline{M}\} \succeq \overline{P}[x := \overline{M}]$$

5. The only non-trivial case is that of a command

$$\langle \widehat{P} \| \bar{\mu}x.\langle M \| E \rangle \rangle$$

This maps to

$$\langle \underline{\overline{M}} \| \underline{\overline{E}} \rangle [x := \underline{\overline{\widehat{P}}}]$$

for which the claim holds by induction.

□

Corollary. *$\bar{\lambda}\bar{\mu}$ is confluent.*

Proof. Given a span $Q \xleftarrow{\preceq} P \xrightarrow{\succeq} Q'$, by confluence of $\bar{\lambda}$ there exists a $\bar{\lambda}$ term M such that $\overline{Q} \xrightarrow{\succeq} M \xleftarrow{\preceq} \overline{Q'}$. We get

$$\begin{tikzcd}
 & Q \arrow[r] & \underline{\overline{Q}} \arrow[dr] & \\
P \arrow[ur] \arrow[dr] & & & \underline{M} \\
 & Q' \arrow[r] & \underline{\overline{Q'}} \arrow[ur] &
\end{tikzcd}$$

□

It was observed by Curien and Herbelin in [CH00] that the stoup in LJT sequents, which contains at most one formula or type, is the *dual* of the restricted antecedent (at most one formula to the right of $\triangleright$) in LJ. This suggests that to extend LJT to classical logic, we should use LJT$_{\bar{\mu}}$ and simply allow more than one covariable on the right-hand side.

2.3 Classical Sequent Calculus

2.3.1 Definition. The classical sequent calculus LK [Gen69] is defined in figure 2.3.1, where Γ, Δ are sequences of formulas. The *structural* rules Ax, Wk, Cn and Cut deal with the form of sequents while the other *logical* rules deal with connectives.

LK$^{\rightarrow}$ is the subsystem of LK that contains the structural rules plus $\rightarrow_L, \rightarrow_R$. The intuitionistic sequent calculus LJ$^{\rightarrow}$ (def. 2.1.3) can be defined as consisting of all LK$^{\rightarrow}$ sequents and inference rules with at most one formula to the right of $\triangleright$.

$$\dfrac{}{A \triangleright A}\,\text{Ax}$$

$$\dfrac{\Gamma \triangleright \Delta}{\Gamma, A \triangleright \Delta}\,\text{Wk}_L \qquad \dfrac{\Gamma \triangleright \Delta}{\Gamma \triangleright A, \Delta}\,\text{Wk}_R$$

$$\dfrac{\Gamma, A, A \triangleright \Delta}{\Gamma, A \triangleright \Delta}\,\text{Cn}_L \qquad \dfrac{\Gamma \triangleright A, A, \Delta}{\Gamma \triangleright A, \Delta}\,\text{Cn}_R$$

$$\dfrac{\Gamma \triangleright A, \Delta \qquad \Gamma, A \triangleright \Delta}{\Gamma \triangleright \Delta}\,\text{Cut}$$

$$\dfrac{\Gamma \triangleright A, \Delta \qquad \Gamma, B \triangleright \Delta}{\Gamma, A \to B \triangleright \Delta}\,\to_L \qquad \dfrac{\Gamma, A \triangleright B, \Delta}{\Gamma \triangleright A \to B, \Delta}\,\to_R$$

$$\dfrac{\Gamma, A \triangleright \Delta}{\Gamma, A \wedge B \triangleright \Delta}\,\wedge_L^1 \qquad \dfrac{\Gamma, B \triangleright \Delta}{\Gamma, A \wedge B \triangleright \Delta}\,\wedge_L^2$$

$$\dfrac{\Gamma \triangleright A, \Delta \qquad \Gamma \triangleright B, \Delta}{\Gamma \triangleright A \wedge B, \Delta}\,\wedge_R$$

$$\dfrac{\Gamma, A \triangleright \Delta \qquad \Gamma, B \triangleright \Delta}{\Gamma, A \vee B \triangleright \Delta}\,\vee_L$$

$$\dfrac{\Gamma \triangleright A, \Delta}{\Gamma \triangleright A \vee B, \Delta}\,\vee_R^1 \qquad \dfrac{\Gamma \triangleright B, \Delta}{\Gamma \triangleright A \vee B, \Delta}\,\vee_R^2$$

$$\dfrac{\Gamma \triangleright A, \Delta}{\Gamma, \neg A \triangleright \Delta}\,\neg_L \qquad \dfrac{\Gamma, A \triangleright \Delta}{\Gamma \triangleright \neg A, \Delta}\,\neg_R$$

Figure 2.3.1: LK

A standard argument (e.g. [TS00, prop. 3.1.7] shows that it is possible to 'push' all instances of Wk up to the leaves of a sequent derivation, so that the system that omits the Wk rules and allows axioms of the form $\Gamma, A \triangleright A, \Delta$ is equivalent to LK ([TS00] calls these systems **G2c** and **G1c** respectively).

The $\bar\lambda\tilde\mu$-calculus is generalized as follows, replacing $\widehat{\cdot}$ with a binder μ, dual to $\tilde\mu$, that turns a 'process' into a term by binding an output channel or continuation variable.

2.3.2 Definition ($\bar\lambda\mu\tilde\mu$-calculus [CH00])**.** We assume countably infinite sets of variables $\{x, y, \ldots\}$ and covariables (def. 2.2.9) $\{\alpha, \beta, \ldots\}$. Expressions are *commands* P, *terms* M and *coterms* E. Variables range over terms, covariables over coterms.

$$\begin{array}{rcl} P & ::= & \langle M \| E \rangle \\ M & ::= & x \mid \lambda x.M \mid \mu\alpha.P \\ E & ::= & \alpha \mid M \cdot E \mid \tilde\mu x.P \end{array}$$

2.3.3 Definition. The system $LK_{\overrightarrow{\mu\tilde\mu}}$ is defined in figure 2.3.2. There are three kinds of sequent, one for each kind of $\bar\lambda\mu\tilde\mu$ expression — terms are typed on the

$$\dfrac{}{\Gamma, x : A \rhd x : A \,|\, \Delta}\,\text{Ax}_R \qquad\qquad \dfrac{}{\Gamma \,|\, \alpha : A \rhd \alpha : A, \Delta}\,\text{Ax}_L$$

$$\dfrac{P : \Gamma \rhd \alpha : A, \Delta}{\Gamma \rhd \mu\alpha.P : A \,|\, \Delta}\,\mu \qquad\qquad \dfrac{P : \Gamma, x : A \rhd \Delta}{\Gamma \,|\, \bar{\mu}x.P : A \rhd \Delta}\,\bar{\mu}$$

$$\dfrac{\Gamma \rhd M : A \,|\, \Delta \qquad \Gamma \,|\, E : A \rhd \Delta}{\langle M \| E \rangle : \Gamma \rhd \Delta}\,\text{Cut}$$

$$\dfrac{\Gamma, x : A \rhd M : B \,|\, \Delta}{\Gamma \rhd \lambda x.M : A \to B \,|\, \Delta}\,\to_R \qquad \dfrac{\Gamma \rhd M : A \,|\, \Delta \qquad \Gamma \,|\, E : B \rhd \Delta}{\Gamma \,|\, M \cdot E : A \to B \rhd \Delta}\,\to_L$$

Figure 2.3.2: $LK^{\to}_{\mu\bar{\mu}}$

right and coterms on the left, while commands, not having any distinguished input or output type, are typed by whole sequents.

2.3.4 Remark. The system $LK^{\to}_{\mu\bar{\mu}}$ eliminates explicit weakening using axioms, as in def. 2.3.1, while the cut rule includes implicit contraction of formulas with the same name (variable or covariable) in Γ and Δ. We can thus define contraction as a cut with an axiom:

$$\dfrac{\Gamma \rhd M : A \,|\, \alpha : A, \Delta}{\langle M \| \alpha \rangle : \Gamma \rhd \alpha : A, \Delta}\,\text{Cn}_R \quad = \quad \dfrac{\Gamma \rhd M : A \,|\, \alpha : A, \Delta \qquad \dfrac{}{\Gamma \,|\, \alpha : A \rhd \alpha : A, \Delta}\,\text{Ax}}{\langle M \| \alpha \rangle : \Gamma \rhd \alpha : A, \Delta}\,\text{Cut}$$

$$\dfrac{\Gamma, x : A \,|\, E : A \rhd \Delta}{\langle x \| E \rangle : \Gamma, x : A \rhd \Delta}\,\text{Cn}_L \quad = \quad \dfrac{\dfrac{}{\Gamma, x : A \rhd x : A \,|\, \Delta}\,\text{Ax} \qquad \Gamma, x : A \,|\, E : A \rhd \Delta}{\langle x \| E \rangle : \Gamma, x : A \rhd \Delta}\,\text{Cut}$$

2.3.5 Definition. The reduction rules of $\bar{\lambda}\mu\bar{\mu}$ are as follows:

$$\begin{aligned} \langle \mu\alpha.P \| E \rangle &\succ P[\alpha := E] \\ \langle M \| \bar{\mu}x.P \rangle &\succ P[x := M] \\ \langle \lambda x.M \| N \cdot E \rangle &\succ \langle M[x := N] \| E \rangle \end{aligned}$$

Alternatively, λ may be defined by means of the following rule.

$$\langle \lambda x.M \| N \cdot E \rangle \succ \langle N \| \bar{\mu}x.M \cdot E \rangle$$

2.3.6 Remark. Notice that the μ–$\bar{\mu}$ pair makes $\bar{\lambda}\mu\bar{\mu}$ fundamentally non-confluent — a simple example is

$$\begin{tikzcd} & \langle \mu\alpha.\langle y \| \beta \rangle \| \bar{\mu}x.\langle z \| \gamma \rangle \rangle \arrow[dl, "\preceq"'] \arrow[dr, "\succeq"] & \\ \langle y \| \beta \rangle & & \langle z \| \gamma \rangle \end{tikzcd}$$

$$\frac{\Gamma \,|\, E : A \;\triangleright\; \Delta}{\Gamma \,|\, \pi_1[E] : A \times B \;\triangleright\; \Delta}\times^1_L \qquad \frac{\Gamma \,|\, F : B \;\triangleright\; \Delta}{\Gamma \,|\, \pi_2[F] : A \times B \;\triangleright\; \Delta}\times^2_L$$

$$\frac{\Gamma \;\triangleright\; M : A \,|\, \Delta \qquad \Gamma \;\triangleright\; N : B \,|\, \Delta}{\Gamma \;\triangleright\; (M, N) : A \times B \,|\, \Delta}\times_R$$

$$\frac{\Gamma \;\triangleright\; M : A \,|\, \Delta}{\Gamma \;\triangleright\; (M)\iota_1 : A + B \,|\, \Delta}+^1_R \qquad \frac{\Gamma \;\triangleright\; N : B \,|\, \Delta}{\Gamma \;\triangleright\; (N)\iota_2 : A + B \,|\, \Delta}+^2_R$$

$$\frac{\Gamma \,|\, E : A \;\triangleright\; \Delta \qquad \Gamma \,|\, F : B \;\triangleright\; \Delta}{\Gamma \,|\, [E, F] : A + B \;\triangleright\; \Delta}+_L$$

$$\frac{\Gamma \,|\, E : A \;\triangleright\; \Delta}{\Gamma \;\triangleright\; [E]^{\perp} : \neg A \,|\, \Delta}\neg_R \qquad \frac{\Gamma \;\triangleright\; M : A \,|\, \Delta}{\Gamma \,|\, (M)^{\perp} \;\triangleright\; \Delta}\neg_L$$

Figure 2.3.3: $LK_{\mu\bar{\mu}}$

Thus, cut-elimination in LK is non-deterministic. This can be overcome, but only at the expense of limiting the number of normal forms, by giving priority to one rule over another. Choosing $\bar{\mu}$ gives call-by-name evaluation, while choosing μ gives call-by-value (see [CH00]).

The obvious choice for equality of terms — the symmetric closure of $\succeq$ — leads to the same collapse as was demonstrated in the corollary to proposition 1.5.10. We may define a non-deterministic choice operator just as above: for commands P, Q not containing α, x free, let $P + Q$ be

$$\begin{tikzcd}
 & \langle \mu\alpha.P \| \bar{\mu}x.Q \rangle \arrow[dl, "\preceq"'] \arrow[dr, "\succeq"] & \\
P & & Q
\end{tikzcd}$$

so that if $=_{\succeq}$ is the closure of $\succeq$ then $P =_{\succeq} Q$. In particular, this means that any two LK proofs of the same sequent are equal. This example is usually attributed to Yves Lafont (see [GLT89, Appendix B]).

2.3.7 Definition ($\bar{\lambda}\mu\bar{\mu}_{\times+\perp}$)**.** We add products, sums and negation to $\bar{\lambda}\mu\bar{\mu}$ by adding the following productions to the grammar of def. 2.3.2 (adapted from [Wad03]):

$$\begin{array}{lll} M & ::= & \ldots\ (M, M) \mid (M)\iota_1 \mid (M)\iota_2 \mid [E]^{\perp} \\ E & ::= & \ldots\ [E, E] \mid \pi_1[E] \mid \pi_2[E] \mid (M)^{\perp} \end{array}$$

2.3.8 Definition ($LK_{\mu\bar{\mu}}$)**.** Terms of $\bar{\lambda}\mu\bar{\mu}_{\times+\perp}$ are typed by the rules of def. 2.3.3 plus those in figure 2.3.3.

2.3.9 Definition. The reduction rules of $\bar{\lambda}\mu\bar{\mu}_{\times+\perp}$ are those of def. 2.3.5 plus the following, for $i = 1, 2$:

$$\begin{array}{rcl} \langle (M_1, M_2) \| \pi_i[E] \rangle & \succ & \langle M_i \| E \rangle \\ \langle (M)\iota_i \| [E_1, E_2] \rangle & \succ & \langle M \| E_i \rangle \\ \langle [E]^{\perp} \| (M)^{\perp} \rangle & \succ & \langle M \| E \rangle \end{array}$$

Note the symmetry of the above rules. In particular, $+$ is now clearly the dual of $\times$, in contrast to the rather messy situation in minimal logic.

2.3.10 Remark. We can now prove the excluded middle $A + \neg A$:

$$
\begin{array}{rll}
\hline
\mid \alpha : A & \rhd \ \alpha : A, \beta : A + \neg A & \text{Ax} \\
\hline
 & \rhd \ [\alpha]^{\perp} : \neg A \mid \alpha : A, \beta : A + \neg A & \neg_R \\
\hline
 & \rhd \ ([\alpha]^{\perp})\iota_2 : A + \neg A \mid \alpha : A, \beta : A + \neg A & +^2_R \\
\hline
\langle ([\alpha]^{\perp})\iota_2 \| \beta \rangle : & \rhd \ \alpha : A, \beta : A + \neg A & \mathrm{Cn}_R \\
\hline
 & \rhd \ \mu\alpha.\langle ([\alpha]^{\perp})\iota_2 \| \beta \rangle : A \mid \beta : A + \neg A & \mu \\
\hline
 & \rhd \ (\mu\alpha.\langle ([\alpha]^{\perp})\iota_2 \| \beta \rangle)\iota_1 : A + \neg A \mid \beta : A + \neg A & +^1_R \\
\hline
\langle (\mu\alpha.\langle ([\alpha]^{\perp})\iota_2 \| \beta \rangle)\iota_1 \| \beta \rangle : & \rhd \ \beta : A + \neg A & \mathrm{Cn}_R \\
\hline
 & \rhd \ \mu\beta.\langle (\mu\alpha.\langle ([\alpha]^{\perp})\iota_2 \| \beta \rangle)\iota_1 \| \beta \rangle : A + \neg A & \mu
\end{array}
$$

The resulting term can be cut against a coterm of the same type, which (if it is cut-free) will be of the form $[E, (M)^{\perp}]$, yielding

$$\langle \mu\beta.\langle (\mu\alpha.\langle ([\alpha]^{\perp})\iota_2 \| \beta \rangle)\iota_1 \| \beta \rangle \| [E, (M)^{\perp}] \rangle$$

Reducing the μ- and ι-redexes gives

$$\langle \mu\alpha.\langle [\alpha]^{\perp} \| (M)^{\perp} \rangle \| E \rangle$$

The μ-redex converts to

$$\langle [E]^{\perp} \| (M)^{\perp} \rangle$$

which finally reduces to

$$\langle M \| E \rangle$$

The intuitive interpretation of the above is this: a proof that depends on the excluded middle has two branches — one for the case that P and another for the case that $\neg P$. From the syntactic form of proofs we know that the second branch must contain a counterexample to $\neg P$, which is a proof of P. The excluded middle simply extracts this proof and supplies it to the first branch.

The symmetry of the typing and reduction rules for $LK_{\mu\bar{\mu}}$, and the duality between terms and coterms, might make us think of process calculi such as π-calculus. A term $\mu\alpha.P$ can be thought of as a process P that sends its result to an output channel α, while a coterm $\bar{\mu}x.P$ is a process that accepts an input x. The 'process constructor' $\langle \, \cdot \, \| \, \cdot \, \rangle$ plugs a term and a coterm together, connecting the output of the first to the input of the second. These ideas are made more explicit in the $\mathcal{X}$-calculus [vBLL05], a graphical notation for classical proofs derived from Lengrand's $\lambda\xi$ (def. 2.4.2). More concrete results in this vein are given in [BBS97], where a non-constructive proof in Peano arithmetic is analysed and shown to behave much as a system of asynchronous communicating processes.

2.4 Variants

The system $\bar\lambda\mu\bar\mu_{\times+\bot}$ is not strictly isomorphic to LK, because of the distinction between terms, coterms and commands. The correspondence can be turned into an isomorphism by preceding each use of an $LK^{\rightarrow}_{\mu\bar\mu}$ rule by a μ or $\bar\mu$, to single out the active formula, and following it by a contraction or Cut, so that each inference of LK is represented by a command.

2.4.1 Definition. The subsyntax of $\bar\lambda\mu\bar\mu$ determined by the above convention is called $\bar\lambda\mu\bar\mu_K$. The implicative fragment is typed as follows.

$$\frac{}{\langle x\|\alpha\rangle : (\Gamma, x : A \rhd \alpha : A, \Delta)}\;\text{Ax}$$

$$\frac{P : (\Gamma \rhd \alpha : A, \Delta) \qquad P' : (\Gamma, x : A \rhd \Delta)}{\langle \mu\alpha.P \| \bar\mu x.P'\rangle : (\Gamma \rhd \Delta)}\;\text{Cut}$$

$$\frac{P : (\Gamma, x : A \rhd \alpha : B, \beta : A \to B, \Delta)}{\langle \lambda x.\mu\alpha.P \| \beta\rangle : (\Gamma \rhd \beta : A \to B, \Delta)}\;\to_R$$

$$\frac{P : (\Gamma, x : A \to B \rhd \alpha : A, \Delta) \qquad P' : (\Gamma, x : A \to B, y : B \rhd \Delta)}{\langle x \| \mu\alpha.P \cdot \bar\mu x.P'\rangle : (\Gamma, x : A \to B \rhd \Delta)}\;\to_L$$

This syntax is a term calculus for the classical variant of Kleene's system G3 (see [TS00, section 3.5]).

2.4.2 Definition. Urban's system $\mathcal{T}$ [Urb00] and Lengrand's system $\lambda\xi$ [Len03, vBL08] are variants of $\bar\lambda\mu\bar\mu_K$ and have the following syntax.

$\bar\lambda\mu\bar\mu_K$	$\mathcal{T}$	$\lambda\xi$
$\langle x\|\alpha\rangle$	$\mathsf{Ax}(x, \alpha)$	$\langle x \cdot \alpha\rangle$
$\langle \mu\alpha.P \| \bar\mu x.P'\rangle$	$\mathsf{Cut}(\langle\alpha\rangle P, (x)P')$	$P\hat\alpha \dagger \hat x P'$
$\langle \lambda x.\mu\alpha.P \| \beta\rangle$	$\mathsf{Imp}_R((x)\langle\alpha\rangle P, \beta)$	$\hat x P \hat\alpha \cdot \beta$
$\langle x \| \mu\alpha.P \cdot \bar\mu y.P'\rangle$	$\mathsf{Imp}_L(\langle\alpha\rangle P, (y)P', x)$	$P\hat\alpha[x]\hat y P'$

Typing and reduction rules for $\mathcal{T}$ and $\lambda\xi$ can be easily inferred from those of $\bar\lambda\mu\bar\mu_K$.

Natural deduction systems for classical logic can be reconstructed from the LK-based systems.

2.4.3 Definition. The $\lambda\mu$-calculus ([Par92]) is a term calculus for multiple-conclusion natural deduction NK_m (def. 1.3.6).

$$\frac{}{\Gamma, x : A \rhd x : A \,|\, \Delta}\;\text{Ax}$$

$$\frac{\Gamma, x : A \rhd M : B \,|\, \Delta}{\Gamma \rhd \lambda x.M : A \to B \,|\, \Delta}\;\to^{+} \qquad \frac{\Gamma \rhd M : A \to B \,|\, \Delta \qquad \Gamma \rhd N : A \,|\, \Delta}{\Gamma \rhd MN : B \,|\, \Delta}\;\to^{-}$$

$$\frac{\Gamma \rhd M : A \,|\, \Delta}{\Gamma \rhd [\alpha]M : \bot \,|\, \alpha : A, \Delta}\;\text{Name} \qquad \frac{\Gamma \rhd M : \bot \,|\, \alpha : A, \Delta}{\Gamma \rhd \mu\alpha.M : A \,|\, \Delta}\;\mu$$

The β-reduction relation is given by

$$\begin{array}{lcl} (\lambda x.M)N & \succ & M[x := N] \\ (\mu\alpha.M)N & \succ & \mu\alpha.M[[\alpha]L := [\alpha](LN)] \end{array}$$

The second rule has the effect of passing a term's context to named subterms, and is thus closely analogous to the μ rule for $\bar{\lambda}\mu\bar{\mu}$.

One disadvantage with $\lambda\mu$ is that in the reduction rule for μ the binder $\mu\alpha$ does not disappear in the reduct. This can be worked around by adding a rule that removes μ at the top level.

2.4.4 Remark. The double-negation rule can be derived in NK_m. The resulting $\lambda\mu$ term is derived as follows.

$$\dfrac{\dfrac{\dfrac{}{y : \neg\neg A \;\rhd\; y : \neg\neg A \mid} \qquad \dfrac{\dfrac{\dfrac{}{x : A \;\rhd\; x : A \mid}}{x : A \;\rhd\; [\alpha]x : \bot \mid \alpha : A}}{\rhd\; \lambda x.[\alpha]x : \neg A \mid \alpha : A}}{\dfrac{y : \neg\neg A \;\rhd\; y\ \lambda x.[\alpha]x : \bot \mid \alpha : A}{y : \neg\neg A \;\rhd\; \mu\alpha.(y\ \lambda x.[\alpha]x) : A \mid}}}{\rhd\; \lambda y.\mu\alpha.(y\ \lambda x.[\alpha]x) : \neg\neg A \to A \mid}$$

Let this term be denoted by C. Applied to arguments $M, N_1, \ldots, N_n$, the reduction rules give

$$\begin{array}{rcl} CMN_1 \cdots N_n & \succeq & (\mu\alpha.(M\ \lambda x.[\alpha]x)N_1 \cdots N_n \\ & \succeq & \mu\alpha.(M\ \lambda x.(xN_1 \cdots N_n)) \end{array}$$

So C takes an argument and applies it, at the top level, to a function which in turn throws its argument to the continuation of the call to C. In effect, C is the (call-by-name variant of the) `call/cc` operator familiar from Scheme and ML, the only major difference being that C escapes to the top level first.

2.4.5 Definition. The (call-by-name) $\lambda\mathcal{C}$-calculus [FH92] is defined over the λ-calculus augmented with constants $\mathcal{A}, \mathcal{C}$, and with the following reduction rules, for K a λ-context (def. 2.2.2).

$$\begin{array}{rcl} [\mathcal{A}M]K & \succ & M \\ [\mathcal{C}M]K & \succ & M(\lambda v.(\mathcal{A}\ [v]K)) \end{array}$$

Observe that $\mathcal{A}$ as defined here has the same effect as the version in remark 1.5.9.

2.4.6 Remark. The first person to notice that classical logic could be given a Curry–Howard-style interpretation was Griffin [Gri90], who observed that the constant $\mathcal{C}$ above could be given the (polymorphic) type $\neg\neg A \to A$, and that similarly $\mathcal{A} : \bot \to A$:

$$\frac{\Gamma \;\rhd\; M : \bot}{\Gamma \;\rhd\; \mathcal{A}_A M : A} \qquad\qquad \frac{\Gamma \;\rhd\; M : \neg\neg A}{\Gamma \;\rhd\; \mathcal{C}_A M : A}$$

Thus $\mathcal{A}$ and $\mathcal{C}$ can be used to interpret proofs in NJ and NK respectively, although there is a slight complication with the typing of $\mathcal{C}$ (see [Gri90, section 3] for details) which means that to evaluate a $\lambda\mathcal{C}$-term M we must in fact start with $\mathcal{C}\lambda k.kM$.

Both $\lambda\mu$ and $\lambda\mathcal{C}$ have cumbersome provisos relating to the handling of terms at the top level, which suggests that sequent calculus is a better system than natural deduction for investigating the computational properties of classical logic.

A more elegant system of classical natural deduction, quite different to the two above, is given in [BB96]. We describe the fragment corresponding to propositional logic.

2.4.7 Definition ($\lambda^{\mathrm{Sym}}_{\mathrm{Prop}}$)**.** The types A of $\lambda^{\mathrm{Sym}}_{\mathrm{Prop}}$ are defined over sets $\mathcal{P} = \{P, Q, R, \ldots\}$ and $\mathcal{P}^{\perp} = \{P^{\perp}, Q^{\perp}, R^{\perp}, \ldots\}$ of atomic and negated atomic types by

$$\begin{array}{rcl} A' & ::= & \mathcal{P} \mid \mathcal{P}^{\perp} \mid A' \wedge A' \mid A' \vee A' \\ A & ::= & A' \mid \perp \end{array}$$

The operation $\cdot^{\perp}$ is extended to formulas A' via the de Morgan rules. The inference rules are

$$x : A$$

$$\frac{M : A \qquad N : B}{\langle A, B\rangle : A \wedge B} \qquad \frac{M : A_i}{\sigma_i M : A_1 \vee A_2}$$

$$\frac{\begin{array}{c} \overline{x : A} \\ \vdots \\ M : \perp \end{array}}{\lambda x.M : A^{\perp}} \qquad \frac{M : A^{\perp} \qquad N : A}{M * N : \perp}$$

The β-reduction rules are

$$\begin{array}{rcl} \lambda x.M * N & \succeq & M[x := N] \\ M * \lambda x.N & \succeq & N[x := M] \\ \\ \langle M_1, M_2\rangle * \sigma_i N & \succeq & M_i * N \\ \sigma_i N * \langle M_1, M_2\rangle & \succeq & N * M_i \end{array}$$

Observe that $\lambda^{\mathrm{Sym}}_{\mathrm{Prop}}$, like $\bar{\lambda}\mu\tilde{\mu}$, is fundamentally non-confluent.

The paper [BB96] also defines $\lambda^{\mathrm{Sym}}_{\mathrm{PA}}$, a term calculus for higher-order Peano arithmetic. In a later paper [BBS97], the authors give a detailed examination of the computational behaviour of a non-constructive proof in the latter system.

2.5 Models

The traditional algebraic models of classical propositional logic are Boolean algebras, i.e. Heyting algebras satisfying $1 \leq a \vee \neg a$ for all a. The denotation of an LK sequent $\Gamma \rhd \Delta$ is an inequality

$$\bigwedge \Gamma \leq \bigvee \Delta$$

On the other hand, we have seen (prop. 1.5.10 and its corollary) that giving a categorical model of classical proofs (as opposed to an algebraic model of classical provability) is non-trivial, because a CCC where negation is an involution collapses to a preorder. Among the models that have been proposed, we may distinguish between two approaches:

- Weakening the axioms of a BCC to avoid the collapse, while adding enough structure to interpret classical proofs. This is the approach taken by Selinger [Sel01], based on earlier work by Filinski [Fil89], Power and Robinson [PR97] and Pym and Ritter [PR01]. An equivalent formulation was given earlier by Ong [Ong96] in terms of fibrations.

- Adding structure to $*$-autonomous categories (models of multiplicative linear logic — see [BW95, BS04]) to interpret classical $\wedge$ and $\vee$. This is done in various ways by [FP06, Str07, Lam07].

The relationships between these proposed models are far from clear, and the current *embarras du choix* is summed up well by Lamarche:

> ... [We] have gone from having no answer at all to having way too many answers. [Lam07, p. 473]

We do not have the space to describe these models in full. Instead we sketch some of their important features. These descriptions are not rigorous — see the above papers for the definitive accounts.

Control Categories

Selinger weakens the structure of BCCs by not requiring the interpretation of $\vee$ to be functorial in both variables simultaneously. Disjunction is interpreted as a *premonoidal* functor $\oplus$, which means roughly that the action of $\oplus$ on two morphisms need not be defined.

2.5.1 Definition ([PR97])**.** A *binoidal category* $\langle \mathbf{C}, F \rangle$ consists of a category $\mathbf{C}$ and a pair of endofunctors (F_1^C, F_2^C) for each $C \in \mathbf{C}$ such that $F_1^A B = F_2^B A$ for all $A, B \in \mathbf{C}$.

Because F_1 and F_2 agree where they are defined, their values may be written as $F(A,B), F(A,f)$ and so on — but the notation $F(g,f)$ is not well defined, since F is not a bifunctor. That is, the following diagram, where $f : A \to B, g : C \to D$, need not commute:

$$\begin{tikzcd}
F(A,C) \arrow[r, "F(f,C)"] \arrow[d, "F(A,g)"'] \arrow[dr, phantom, "+"] & F(B,C) \arrow[d, "F(B,g)"] \\
F(A,D) \arrow[r, "F(f,D)"'] & F(B,D)
\end{tikzcd}$$

In fact, commutativity of this diagram is the definition of bifunctoriality for F.

A premonoidal category is just like a monoidal category, except that the tensor product need only be binoidal (not bifunctorial).

2.5.2 Definition. A *strict premonoidal category* consists of a binoidal category $\langle \mathbf{C}, \otimes \rangle$ together with an object I of $\mathbf{C}$ such that for all $A, B, C \in \mathbf{C}$:

$$\begin{array}{rcl} A \otimes (B \otimes C) & = & (A \otimes B) \otimes C \\ A \otimes I & = & A \\ I \otimes A & = & A \end{array}$$

A premonoidal category is the same, except the equations above are replaced with natural isomorphisms a, r, l such that the usual diagrams (the Mac Lane pentagon and triangle) commute (see e.g. [Mac98, chapter VII]). Symmetric premonoidal structure is defined similarly.

The tensor in a premonoidal category is intended to model the combination of computations with side-effects, where composing two computations f and g in parallel does not make sense, because the two compositions in the diagram above may have different effects.

To interpret classical logic, we need a CCC together with an interpretation of $\vee$ that has suitable properties.

2.5.3 Definition ([Sel01])**.** Let $\mathbf{C}$ be a CCC with a symmetric premonoidal functor $\oplus$. $\mathbf{C}$ is a *control category* if the following hold:

1. $\mathbf{C}$ is distributive, i.e. for all objects A the functor $A \oplus -$ preserves finite products.

2. For each object A there is a chosen symmetric $\oplus$-monoid structure [Mac98, section 7.3] $\langle A, \eta, \mu \rangle$, i.e. two maps $\eta : I \to A$ (unit) and $\mu : A \oplus A \to A$ (multiplication) satisfying the same equations as for a symmetric monoid in a symmetric monoidal category:

   $$\mu(\mu \oplus A) = \mu(A \oplus \mu)a \qquad \mu(\eta \oplus A) = l \qquad \mu(A \oplus \eta) = r \qquad \mu s = \mu$$

   where a, l, r, s are the associator, left and right unit and symmetry isomorphisms.

3. The canonical map $s_{ABC} : (A \Rightarrow B) \oplus C \to A \Rightarrow (B \oplus C)$ is a natural isomorphism.

See [Sel01] for the coherence conditions that apply to the above.

Condition 2 is necessary to model right weakening (composition with η) and contraction (composition with μ). Condition 3 allows us to model the $\to^+$ rule, via

$$\frac{\dfrac{f : \Gamma \times B \to A \oplus \Delta}{\phi(f) : \Gamma \to B \Rightarrow (A \oplus \Delta)}}{\tilde{s} \circ \phi(f) : \Gamma \to (B \Rightarrow A) \oplus \Delta}$$

Control categories are models of call-by-name $\lambda\mu$-calculus with ‘classical disjunction’ (see [PR01]). Interestingly, their duals, the co-control categories, model the call-by-value calculus. This duality between the two evaluation strategies was first noticed by Filinski [Fil89]. A disadvantage of this approach is that it requires a choice of evaluation strategy once and for all (cf. remark 2.3.6).

Extensions of Linear Logic

Syntactically, classical logic may be recovered from linear logic by readmitting weakening and contraction. However, things are not so simple on the semantic side, as we will see. Models of multiplicative linear logic are $*$-autonomous categories, or equivalently (symmetric) linearly distributive categories with negation (see [CS97]).

2.5.4 Definition ([CS97])**.** A *linearly distributive category* (LDC, also called a 'weakly distributive category') is a category $\mathbf{C}$ equipped with two monoidal structures $\langle \otimes, \top \rangle$ and $\langle \oplus, \bot \rangle$ and two natural transformations:

$$\begin{array}{rcl} d_L^L & : & A \otimes (B \oplus C) \to (A \otimes B) \oplus C \\ d_R^R & : & (B \oplus C) \otimes A \to B \oplus (C \otimes A) \end{array}$$

subject to the coherence conditions given in [CS97].

A *symmetric* LDC is an LDC where the two monoidal functors are symmetric. An SLDC *has negation* if to each object A there is associated a chosen dual object A^* and two maps

$$\begin{array}{rcl} \gamma_A & : & A \otimes A^* \to \bot \\ \tau_A & : & \top \to A \oplus A^* \end{array}$$

again satisfying some coherence conditions.

An SLDC with negation is equivalent to a $*$-autonomous category, as shown in [CS97]. In particular, the object map $\cdot^*$ extends to a contravariant endofunctor that interprets negation, and there is a linear implication $A \multimap B$ defined as $A^* \oplus B$. A sequent $\Gamma \,\triangleright\, \Delta$ is interpreted as a map $\bigotimes \Gamma \to \bigoplus \Delta$, and the distributivity maps are used to interpret the cut rule: given $f : \Gamma \to \Delta \oplus A$ and $g : A \otimes \Gamma' \to \Delta'$, we may form

$$\Gamma \otimes \Gamma' \xrightarrow{f \otimes \Gamma'} (\Delta \oplus A) \otimes \Gamma' \xrightarrow{d} \Delta \oplus (A \otimes \Gamma') \xrightarrow{\Delta \oplus g} \Delta \oplus \Delta'$$

Naïvely, then, one might expect that to interpret weakening and contraction, and thus classical logic, we need only choose symmetric $\otimes$-comonoid[6] and $\oplus$-monoid structures for each object and require every morphism to commute with these. Such a model, however, is subject to Joyal's collapse, as a result of the following.

2.5.5 Proposition. *Given a symmetric monoidal category $\langle \mathbf{C}, \otimes, \top \rangle$, if every object A of $\mathbf{C}$ is equipped with a chosen symmetric $\otimes$-comonoid structure $\langle A, \epsilon_A : A \to \top, \delta_A : A \to A \otimes A \rangle$, and if every morphism in $\mathbf{C}$ is a comonoid morphism, then the monoidal structure on $\mathbf{C}$ is cartesian product.* [7]

[6] A (symmetric) comonoid object is dual to a monoid object: it is an object A with maps $\epsilon : A \to \top, \delta : A \to A \otimes A$ satisfying

$$(A \otimes \delta)\delta = a(\delta \otimes A)\delta \qquad l(A \otimes \epsilon)\delta = 1 \qquad r(\epsilon \otimes A)\delta = 1 \qquad (s\delta = \delta)$$

[7] This result and its dual are mentioned in the literature (see e.g. [Lam07, p. 473], [Mac65, p. 80]), but we have been unable to find a proof, and so present our own.

Proof. We exhibit $\otimes$ as right adjoint to the diagonal functor $\Delta : \mathbf{C} \to \mathbf{C} \times \mathbf{C}$, with unit and counit (calling the counit ζ because ϵ is already in use)

$$\begin{aligned} \eta_A : A \to A \otimes A &= \delta_A \\ \zeta_{(A,B)} : (A \otimes B, A \otimes B) \to (A, B) &= (r(A \otimes \epsilon_B), l(\epsilon_A \otimes B)) \end{aligned}$$

where $(\,\cdot\,,\,\cdot\,)$ denotes objects and arrows in the product category $\mathbf{C} \times \mathbf{C}$. These are natural because every arrow in $\mathbf{C}$ is a comonoid morphism, meaning that δ_A and ϵ_A are natural in A. The triangle identities are

$$\begin{aligned} (r(A \otimes \epsilon_A), l(\epsilon_A \otimes A)) \circ (\delta_A, \delta_A) &= (1_A, 1_A) \\ (r(A \otimes \epsilon_B) \otimes l(\epsilon_A \otimes B)) \circ \delta_{A \otimes B} &= 1_{A \otimes B} \end{aligned}$$

The first follows immediately from the definition of a comonoid (footnote 6) and composition in a product category. For the second, consider the maps

$$\gamma_{AB} = m(\delta_A \otimes \delta_B) \qquad \beta_{AB} = r_\top(\epsilon_A \otimes \epsilon_B)$$

where $m : (A \otimes A) \otimes (B \otimes B) \to (A \otimes B) \otimes (A \otimes B)$ is the unique isomorphism given by coherence. These define a comonoid structure on $A \otimes B$, as is shown by naturality of m, r and coherence. We show that in this situation comonoid structures are unique, and hence that $\gamma_{AB} = \delta_{A \otimes B}$.

First note that $\epsilon_\top = 1_\top$, since by naturality of ϵ we have $\epsilon_\top = \epsilon_\top \epsilon_\top$, and by naturality of l

$$\begin{aligned} 1_\top &= l(\epsilon_\top \otimes \top)\delta_\top \\ &= l(\epsilon_\top \otimes \top)(\epsilon_\top \otimes \top)\delta_\top \\ &= \epsilon_\top l(\epsilon_\top \otimes \top)\delta_\top \\ &= \epsilon_\top \end{aligned}$$

Let $\langle f : A \to A^2, e : A \to \top \rangle$ be a comonoid on A. Naturality of ϵ gives $\epsilon_A = \epsilon_\top e = e$, so ϵ is unique. Hence $\top$ is a terminal object, and in particular $\epsilon_{A \otimes B} = r_\top(\epsilon_A \otimes \epsilon_B)$ as above.

Now consider the following diagram, in which exponents denote tensor powers (e.g. $f^2 = f \otimes f$).

$$\begin{tikzcd}
A^2 \arrow[rrdd, "\delta_{A^2}"] & & & & A^2 \otimes \top \arrow[rrdd, "r_{A^2}"] & & \\
& & & (2) & & & \\
A \arrow[uu, "f"] \arrow[dd, "\delta_A"'] & (1) & A^4 \arrow[rruu, "A^2 \otimes \epsilon_{A^2}"] \arrow[rr, "A^2 \otimes \epsilon_A^2"] \arrow[rrdd, "(A \otimes \epsilon_A)^2"'] & & A^2 \otimes \top^2 \arrow[uu, "A^2 \otimes r_\top"'] \arrow[dd, "m"] & (4) & A^2 \\
& & & (3) & & & \\
A^2 \arrow[rruu, "f^2"'] & & & & (A \otimes \top)^2 \arrow[rruu, "r_A^2"'] & &
\end{tikzcd}$$

The components commute for the following reasons.

(1). By naturality of δ.

(2). By uniqueness of ϵ.

(3). By naturality of m, coupled with $1_{A^4} = m : A^4 \to A^4$ by coherence.

(4). By coherence.

The entire diagram thus commutes, while the comonoid laws and the uniqueness of ϵ show that the top and bottom paths equal f and δ_A respectively. It follows that δ_A is the unique comultiplication on A for any A, and thus that $\gamma_{AB} = m(\delta_A \otimes \delta_B)$ above is equal to $\delta_{A\otimes B}$. Now we have

$$\begin{aligned}
& (r(A\otimes\epsilon_B)\otimes l(\epsilon_A\otimes B))\delta_{A\otimes B} \\
= \;& (r\otimes l)(A\otimes\epsilon_B\otimes\epsilon_A\otimes B)m(\delta_A\otimes\delta_B) \\
= \;& (r\otimes l)(A\otimes\epsilon_A\otimes\epsilon_B\otimes B)(\delta_A\otimes\delta_B) \\
= \;& r(A\otimes\epsilon_A)\delta_A\otimes l(\epsilon_B\otimes B)\delta_B \\
= \;& 1_{A\otimes B}
\end{aligned}$$

Hence $\Delta \dashv \otimes$, and so $\otimes$ is cartesian product. □

If $\mathbf{C}$ is $*$-autonomous, it is closed and has an involutive negation, and so prop. 1.5.10 applies, showing that $\mathbf{C}$ is a preorder. The solution is not to require each morphism in $\mathbf{C}$ to commute with the comonoid (monoid) structures — that is, δ_A (μ_A) and ϵ_A (η_A) need not be natural in A. We mention two references:

1. Führmann and Pym's classical categories [FP06] are poset-enriched SLDCs with negation and symmetric (co)monoids, where the (co)unit and (co)multiplication are required to be *lax* natural, i.e. we should have $f\circ\eta_A \leq \eta_B$, $f\circ\mu_A \leq \mu_B\circ(f\otimes f)$, and so on. Here $f \leq g$ expresses that f reduces to g under cut elimination. (Cf. remark 1.4.7.)

2. Straßburger's $\mathsf{B1}$-categories [Str07] are $*$-autonomous categories with (symmetric) $\oplus$-monoids and $\otimes$-comonoids, but without any commutativity conditions on morphisms. The paper goes on to examine in depth various such conditions and their consequences.

Despite the current confusion, there is a definite consensus that linear logic and $*$-autonomous categories are the better foundation for a semantics of classical proofs.

2.6 Conclusions

We have only scratched the surface of the Curry–Howard correspondence: for example, we have not even mentioned predicate logic or inductive types, or the 'negative translations' of classical into intuitionistic logic and their connections with CPS transforms [Gri90] and monads in functional programming [HD94]. Nevertheless, the reader should be convinced that the correspondence is at the very least a useful analogy, and that the study of proof theory may be fruitfully applied to the problem of deriving provably correct programs as well as in general theories of computation. In particular, the fact that classical logic presents such familiar problems as that of giving a suitable notion of equality for non-deterministic rewrite systems indicates that something non-trivial is going on.

Conversely, it may be that the coinductive methods of operational semantics and concurrency theory will shed light on this logical question.

Indeed, the BHK interpretation (and realizability semantics in general) can be seen as a particularly simple operational semantics of proofs, although because only the output of a realizer matters, this (largely informal) semantics coincides with the denotational kind. In the world of classical logic, there are interesting similarities between Abramsky's 'process realizability' for linear logic using CCS [Abr00] and Barbanera and Berardi's 'symmetric reducibility candidates' for classical logic [BB96] as formalized in [DGLL05]. Perhaps these will be useful in extending to classical logic the work already done on the operational semantics of linear proofs.

We may sum up by saying that the question of what a classical proof 'means' is in no way settled, and that in answering it we will need the methods of both logic and computer science.

References

[Abr93] Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111:3–57, 1993.

[Abr00] Samson Abramsky. Process realizability. In F. L. Bauer and R. Steinbruggen, editors, *Foundations of Secure Computation: Proceedings of the 1999 Marktoberdorf Summer School*, pages 167–180. IOS Press, 2000.

[AR01] Jeremy Avigad and Erich H. Reck. 'Clarifying the nature of the infinite': the development of metamathematics and proof theory. Technical Report CMU-PHIL-120, Carnegie-Mellon University, 2001.

[Bar81] Henk Barendregt. *The λ-Calculus: Its Syntax and Semantics.* North-Holland, 1981.

[BB96] Franco Barbanera and Stefano Berardi. A symmetric lambda calculus for classical program extraction. *Information and Computation*, 125:103–117, 1996.

[BBS97] Franco Barbanera, Stefano Berardi, and Massimo Schivalocchi. "Classical" programming-with-proofs in $\lambda_{\mathrm{PA}}^{\mathrm{Sym}}$: an analysis of non-confluence. In *Proc. TACS '97*, volume 1281 of *LNCS*, 1997.

[BS04] Richard Blute and Philip Scott. Category theory for linear logicians. In Thomas Ehrhard, Jean-Yves Girard, Paul Ruet, and Philip Scott, editors, *Linear Logic in Computer Science*, volume 316 of *LMS Lecture Notes*, pages 3–65. Cambridge University Press, 2004.

[BW95] Michael Barr and Charles Wells. *Category Theory for Computing Science.* Prentice-Hall, second edition, 1995.

[CF68] Haskell B. Curry and Robert Feys. *Combinatory Logic*, volume 1. North-Holland, 1968.

[CH00] Pierre-Louis Curien and Hugo Herbelin. The duality of computation. *ACM SIGPLAN Notices*, 35(9):233–243, 2000.

[CS97] J. R. B. Cockett and R. A. G. Seely. Weakly distributive categories. *Journal of Pure and Applied Algebra*, 114:133–173, 1997.

[DGLL05] Daniel Dougherty, Silvia Ghilezan, Pierre Lescanne, and Silvia Likavec. Strong normalization of the dual classical sequent calculus. In *Proc. LPAR '05*, volume 3835 of *LNCS*, pages 169–183. Springer-Verlag, 2005.

[Dum00] Michael Dummett. *Elements of Intuitionism.* Number 39 in Oxford Logic Guides. Oxford University Press, second edition, 2000.

[FH92] Matthias Felleisen and Robert Hieb. A revised report on the syntactic theories of sequential control and state. *Theoretical Computer Science*, 103(2):235–271, 1992.

[Fil89] Andrzej Filinski. Declarative continuations and categorical duality. Master's thesis, University of Copenhagen, 1989.

[FP06] Carsten Führmann and David Pym. Order-enriched categorical models of the classical sequent calculus. *Journal of Pure and Applied Algebra*, 204:21–78, 2006.

[Gal93] Jean Gallier. Constructive logics, part I: A tutorial on proof systems and typed λ-calculi. *Theoretical Computer Science*, 110(2):249–339, 1993.

[Gen69] Gerhard Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The Collected Works of Gerhard Gentzen*, chapter 3. North-Holland, 1969.

[Gha95] Neil Ghani. *Adjoint Rewriting.* PhD thesis, University of Edinburgh, 1995.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[GLT89] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types.* Cambridge University Press, 1989.

[Gri90] Timothy Griffin. A formulae-as-types notion of control. In *Conf. Record 17th Annual ACM Symp. on Principles of Programming Languages.* ACM Press, 1990.

[HD94] John Hatcliff and Olivier Danvy. A generic account of continuation passing styles. In *Proc. POPL '94.* ACM Press, 1994.

[Her94] Hugo Herbelin. A λ-calculus structure isomorphic to Gentzen-style sequent calculus structure. In *Proc. Computer Science Logic '94*, volume 933 of *LNCS*, pages 61–75. Springer-Verlag, 1994.

[Her05] Hugo Herbelin. C'est maintenant qu'on calcule: au cœur de la dualité. Habilitation à diriger les recherches, Université Paris 11, 2005. `http://pauillac.inria.fr/~herbelin/publis/memoire.ps`.

[Hey66] Arend Heyting. *Intuitionism: An Introduction.* North-Holland, second edition, 1966.

[Hil96] Barnaby P. Hilken. Towards a proof theory of rewriting: the simply-typed 2λ-calculus. *Theoretical Computer Science*, 170:407–444, 1996.

[How80] William A. Howard. The formulae-as-types notion of construction. In J. R. Hindley and J. P. Seldin, editors, *To H. B. Curry: Essays on combinatory logic, λ-calculus and formalism.* Academic Press, 1980.

[HS86] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and the λ-Calculus.* Cambridge University Press, 1986.

[Kle45] Stephen Kleene. On the interpretation of intuitionistic number theory. *Journal of Symbolic Logic*, 10(4):109–124, 1945.

[Kol67] A. N. Kolmogorov. On the principle of excluded middle. In Jean van Heijenoort, editor, *From Frege to Gödel*, pages 414–437. Harvard University Press, 1967.

[Lam07] François Lamarche. Exploring the gap between linear and classical logic. *Theory and Applications of Categories*, 18(17):471–533, 2007.

[Len03] Stéphane Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. *Electronic Notes in Theoretical Computer Science*, 86(4):714–730, 2003.

[LS86] J. Lambek and P. J. Scott. *Introduction to Higher-Order Categorical Logic.* Cambridge University Press, 1986.

[Mac65] Saunders Mac Lane. Categorical algebra. *Bulletin of the AMS*, 71:40–106, 1965.

[Mac98] Saunders Mac Lane. *Categories for the Working Mathematician.* Springer-Verlag, second edition, 1998.

[Ong96] C.-H. Luke Ong. A semantic view of classical proofs. In *Proc. LICS '96.* IEEE, 1996.

[Par92] Michel Parigot. $\lambda\mu$-calculus: An algorithmic interpretation of classical natural deduction. In *Proc. Logic Programming and Automated Reasoning '92*, volume 624 of *LNCS*. Springer-Verlag, 1992.

[PR97] John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 7:453–468, 1997.

[PR01] David Pym and Eike Ritter. On the semantics of classical disjunction. *Journal of Pure and Applied Algebra*, 159:315–338, 2001.

[Pra06] Dag Prawitz. *Natural Deduction: A Proof-Theoretical Study.* Dover, 2006.

[See87] R. A. G. Seely. Modelling computations: a 2-categorical framework. In *Proc. Logic in Computer Science '87.* IEEE Press, 1987.

[Sel01] Peter Selinger. Control categories and duality: on the categorical semantics of the $\lambda\mu$-calculus. *Mathematical Structures in Computer Science*, 11:207–260, 2001.

[Str07] Lutz Straßburger. On the axiomatization of Boolean categories with and without medial. *Theory and Applications of Categories*, 18(18):536–601, 2007.

[TS00] Anne Troelstra and Helmut Schwichtenberg. *Basic Proof Theory.* Number 43 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, second edition, 2000.

[Urb00] Christian Urban. *Classical Logic and Computation.* PhD thesis, University of Cambridge, 2000.

[vBL08] Steffen van Bakel and Pierre Lescanne. Computation with classical sequents. *Mathematical Structures in Computer Science*, 18(3):555–609, 2008.

[vBLL05] Steffen van Bakel, Stéphane Lengrand, and Pierre Lescanne. The language $\mathcal{X}$: circuits, computations and classical logic. Technical Report RR2005–11, ENS Lyon, 2005.

[Wad93] Philip Wadler. A taste of linear logic. In *Proc. Mathematical Foundations of Computer Science*, volume 711 of *LNCS*. Springer-Verlag, 1993.

[Wad03] Philip Wadler. Call-by-value is dual to call-by-name. In *Proc. ICFP '03.* ACM, 2003.